Machine Learning Approaches to Refining Post-translational Modification Predictions and Protein Identifications from Tandem Mass Spectrometry

by

Clement Chung

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Copyright © 2012 by Clement Chung
Abstract

Machine Learning Approaches to Refining Post-translational Modification Predictions and Protein Identifications from Tandem Mass Spectrometry

Clement Chung
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2012

Tandem mass spectrometry (MS/MS) is the dominant approach for large-scale peptide sequencing in high-throughput proteomic profiling studies. The computational analysis of MS/MS spectra involves the identification of peptides from experimental spectra, especially those with post-translational modifications (PTMs), as well as the inference of protein composition based on the putative identified peptides. In this thesis, we tackled two major challenges associated with an MS/MS analysis: 1) the refinement of PTM predictions from MS/MS spectra and 2) the inference of protein composition based on peptide predictions. We proposed two PTM prediction refinement algorithms, PTM-Clust and its Bayesian nonparametric extension iPTMClust, and a protein identification algorithm, pro-HAP, that is based on a novel two-layer hierarchical clustering approach that leverages prior knowledge about protein function. Individually, we show that our two PTM refinement algorithms outperform the state-of-the-art algorithms and our protein identification algorithm performs at par with the state of the art. Collectively, as a demonstration of our end-to-end MS/MS computational analysis of a human chromatin protein complex study, we show that our analysis pipeline can find high confidence putative novel protein complex members. Moreover, it can provide valuable insights into the formation and regulation of protein complexes by detailing the specificity of different PTMs for the members in each complex.
Dedication

To my loving parents and wife, Carmen.
Acknowledgements

First, I would like to acknowledge my family, especially my parents and wife, Carmen. I agree with the sentiment that graduate school is a marathon and not a sprint. A PhD degree, in particular, is a long endeavour that has its ups and downs with unexpected twists and turns. My family’s unwavering love and support have helped me stay focused throughout the journey and enabled me to cross the finish line.

A key element to the success of a marathon racer is having proper coaching. I am fortunate to have learned from some of the best researchers and teachers during my graduate study. I would like to thank my advisor, Dr. Brendan Frey, and Doctoral Supervisory committee member, Dr. Andrew Emili, for their limitless patience and guidance. They have instilled in me a passion for research and inspired me to be the best that I can be. I would also like to thank my other Doctoral Supervisory committee member, Dr. Michael Brudno, for his valuable insights and feedbacks on my work. Lastly, I would like to recognize Dr. Anitha Kannan and Dr. Ariel Fuxman from Microsoft Research for taking me under their wings as a research intern. I have learned a lot during the internship. Without the first-rate coaching that I have received throughout the years, I would not have been able to cross the finish line.

In addition, I would like to thank members of the PSI lab and the Emili Lab, past and present, for the many stimulating discussions. I would like to single out Vincent, Jian, Ruth from the Emili Lab for helping to make the collaborative effects in my projects seamless. Furthermore, I would like to thank Babak, Leo, Boyko and Chris from the PSI lab and fellow computer science graduate student Maria for proof-reading my thesis. Finally, for those fellow marathon runners, keep your focus, continue to work hard towards to your goal and enjoy the run. I look forward to celebrating with you when you cross the finish line.
# Contents

**List of Tables** ix

**List of Figures** x

1 **Introduction** 1

1.1 Organization of the Thesis 5

2 **Background on Proteomics Analysis** 6

2.1 Bottom-up Analysis of Protein Mixture using Tandem Mass Spectrometry 9

2.2 Mapping Spectra to Peptides 15

2.2.1 Database Search Algorithms 15

2.2.2 De Novo Sequencing Algorithms 19

2.2.3 Sequence Tag Algorithms 20

2.2.4 Spectra Matching Algorithms 21

2.2.5 Scoring Peptide Predictions 22

2.3 Predicting Peptides with PTMs 24

2.4 Refinement of PTM Predictions 27

2.5 Scoring Predicted PTM Sites 29

2.6 Mapping Peptides to Proteins 30

2.6.1 ProteinProphet 31
3 Background on Machine Learning Concepts

3.1 Graphical Representation of Probabilistic Models

3.1.1 Bayesian Networks

3.1.2 Factor Graphs

3.1.3 Plate Notation

3.2 Machine Learning (ML) Algorithms

3.2.1 Clustering

3.2.2 Example: \( k \)-means Clustering

3.2.3 Example: \( k \)-medoids Clustering

3.2.4 ML and MAP Estimations

3.2.5 The Expectation-Maximization (EM) Algorithm

3.2.6 Example: Mixture of Gaussians (MOG)

3.2.7 Inference with Message Passing Algorithms

- Sum-product Algorithm
- Max-product Algorithm
- Loopy Belief Propagation

3.2.8 Example: Affinity Propagation (AP)

3.2.9 Model Complexity and Model Selection

3.2.10 Automatic Model Selection with Nonparametric Bayesian Models

- Chinese Restaurant Process (CRP)

3.2.11 Inference by Sampling

- Inference with Gibbs Sampling
- Inference with Split-merge Metropolis-Hastings Algorithm

3.2.12 ML Approaches to PTM Refinement and Protein Identification
4 Computational Refinement of PTM Predictions

4.1 Introduction ................................................. 94
4.2 PTMClust Algorithm ...................................... 99
4.3 Experimental Evaluations ................................. 110
  4.3.1 Algorithm Comparison on Synthetic Data .......... 112
  4.3.2 Benchmarking Against Phosphopeptide Predictions 114
  4.3.3 Large-scale PTM Analysis of Yeast Proteome .... 120
4.4 Summary and Future Directions .......................... 123

5 Nonparametric Bayesian Approach to PTM Refinement 127

5.1 Motivation and Background .............................. 129
5.2 Methods ...................................................... 131
  5.2.1 iPTMClust Algorithm ................................. 131
  5.2.2 Inference Method .................................... 141
  5.2.3 Background Model .................................. 143
5.3 Experimental Evaluations ............................... 144
  5.3.1 Algorithm Comparison on Synthetic PTM Data .... 144
  5.3.2 Benchmarking Against Phosphopeptide Predictions 148
  5.3.3 Large-scale PTM Analysis of Yeast Proteome .... 153
  5.3.4 Analysis of Human Protein-protein Interaction Data 156
  5.3.5 Analysis of Number of Samples and Burn-in Period Required 159
5.4 Discussion and Future Directions ....................... 162

6 Protein Identification by Hierarchical Affinity Propagation 167

6.1 Current State of Computational Protein Identification by MS/MS . . 168
6.2 Protein Identification as a Hierarchical Clustering Problem .......... 171
  6.2.1 Protein Identification by FL .......................... 173
  6.2.2 pro-HAP: Protein Identification with Prior Knowledge ....... 174
6.3 Experiments and Results

6.3.1 Evaluation of HAP on Synthetic 2D Data

6.3.2 Analysis of Synthetic HIV Sequences

6.3.3 Benchmark Analysis of Yeast Proteome

6.3.4 Benchmark Analysis of Human Protein-protein Interaction Data

6.3.5 In-depth Analysis of Human Protein-protein Interaction Data

6.4 Summary and Future Directions

7 Concluding Remarks

A Derivation of a Mixture of Gaussians (MOG) Model

B Derivation of PTMClust

C Extending AP to Solve FL Problems

D Derivation of HAP from AP

D.1 Inference by Message Passing for HAP

E SIMS Algorithm

Bibliography
List of Tables

4.1 Benchmark results between PTMClust, SIMS, InsPecT, MODmap and PTMFinder .......................................................... 118
4.2 Summary of known modifications in the yeast proteome dataset ................................................................. 122
5.1 Summary of known PTM sites in the yeast proteome dataset ................................................................. 154
5.2 Summary of peptides with known PTM sites in the yeast proteome dataset .................................................. 155
5.3 Listing of unique putative phosphorylated and acetylated peptides for the MED, POL2 and PRC1 complex ................................................................. 157
6.1 Table of selected identified proteins with associated putative PTMs .......................................................... 189
# List of Figures

1. Mass spectrometry (MS) shotgun proteomics experimental pipeline. . . . 2

2. Protein biosynthesis process with post-translational modification (PTM). 7

2. Components of a tandem ion trap mass spectrometer. . . . . . . . . . . . 10

2.3 An overview of the structure of a peptide, and the b- and y-ion that are
formed by fragmentation along the peptide amine bond. . . . . . . . . . . . 13

2.4 Example of a mass fragmentation ladder. . . . . . . . . . . . . . . . . . . 14

2.5 Theoretical example of a peptide prediction using the basic approach of a
database search algorithm on a noiseless spectrum containing only b-ions. 16

2.6 Theoretical example of a peptide with a PTM predicted using a database
search algorithm on a noiseless spectrum containing only b-ions. . . . . 25

3.1 Examples of a Bayesian network and its corresponding factor graph. . . 41

3.2 Example of a Bayesian network drawn up explicitly and in plate notation. 47

3.3 Illustration showing the result of applying the k-means clustering algorithm. 53

3.4 Illustration showing the result of applying the k-medoids clustering algorithm 55

3.5 Bayesian network for the MOG model. . . . . . . . . . . . . . . . . . . . 61

3.6 Illustration showing the result of applying a mixtures of Gaussians (MOG)
clustering algorithm. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63

3.7 The sum-product (or max-product) algorithm messages illustrated on a
factor graph. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
3.8 Graphical model for Affinity Propagation (AP) drawn as a factor graph.

3.9 Illustration showing the results of applying AP.

3.10 Bayesian networks of a Bayesian finite MOG, an IMM using a DP prior and simplified IMM with a DP prior.

4.1 Histograms of inputs to PTMClust of a phosphopeptide dataset.

4.2 A Bayesian network describing PTMClust, using plate notation.

4.3 Distribution of modified peptides in the yeast proteome dataset with more than three instances.

4.4 Distribution of modification position error used by PTMClust.

4.5 Histograms of synthetic data modification position error.

4.6 Histograms of synthetic data modification mass.

4.7 Clustering algorithm comparison on a synthetically generated dataset.

4.8 Phosphopeptide benchmark experiment flowchart.

5.1 A Bayesian network describing the generative model for our new algorithm iPTMClust, using plate notation.

5.2 Benchmark comparison of clustering algorithms on a synthetic dataset.

5.3 Evaluation of iPTMClust and PTMClust on a synthetic dataset.

5.4 Plots of the number of correct phosphorylation sites identified v.s. the number of false phosphorylation sites identified for iPTMClust, PTM-Clust, InsPecT and PTMFinder on a phosphopeptide dataset with known modification sites.

5.5 Plots of the number of correct phosphorylation sites identified v.s. the number of false phosphorylation sites identified for iPTMClust, PTMClust and SIMS on a phosphopeptide dataset with known modification sites.
5.6 Trace plots of the log joint probability distribution and the number of non-background PTM groups across the MCMC samples for the phosphorylation dataset. 160

5.7 Trace plots of the log joint probability distribution and the number of non-background PTM groups across the MCMC samples for the yeast proteome and human protein complex datasets. 161

6.1 2D synthetic data: comparison between HAP and its greedy counterpart (Greedy). 177

6.2 Synthetic HIV data: precision-recall for HAP, Greedy, HKMC and HK-Means applied to the problem of identifying ancestral sequences from a set of 867 synthetic HIV sequences. 180

6.3 Synthetic HIV data: distribution of Rand index for different experiments using HAP and Greedy. 181

6.4 A plot of the true positive rate (TPR) vs false positive rate (FPR) for pro-HAP, pro-FL, ProteinProphet, MSpresso and MSnet on the problem of protein identification using mass spectrometry data derived from yeast wild-type grown in a rich medium. 183

6.5 Human Protein-protein Interaction Data: distribution of fraction of protein complex members found to decoy protein detected. 185

6.6 Human Protein-protein Interaction Data: distribution of fraction of protein complex members found to decoy protein detected for pro-HAP and pro-FL. 187

D.1 Graphical representation of HAP. 208
Chapter 1

Introduction

Early detection of many life-threatening illnesses, such as cancers and heart disease, is known to greatly increase the success rate of subsequent treatments. Thus, one major focus in current medical research is biomarker discovery: e.g., identification and quantification of proteins that are differentially expressed in diseased individuals. A prominent high-throughput method used in protein biomarker discovery, and more generally in qualitative proteomics, is shotgun proteomics followed by tandem mass spectrometry (MS/MS) [1, 2]. MS/MS is a process in which proteins of unknown identities are broken into peptide fragments, and their identities are inferred computationally by analyzing the experimental mass spectra (hereon referred to as spectra for simplicity). An overview of a typical MS/MS experiment is outlined in Fig. 1.1. The high sensitivity of current mass spectrometers coupled with extensive biochemical fractionation allows several thousand of proteins to be identified at once [3, 4].

Although a large number of proteins can be identified per run of a mass spectrometry (MS) experiment, the yield is far from complete and is biased towards proteins with high abundance. Combining results from technical replications of the same experiment can incrementally improve protein coverage up to a saturation point, but many low abundance proteins may still be undetected [3]. Furthermore, errors and deficiencies arising
Figure 1.1: Mass spectrometry (MS) shotgun proteomics experimental pipeline. The top half illustrates an overview of an experimental procedure for generating tandem mass spectrometry (MS/MS) spectra. It starts with a sample containing an unknown composition of proteins, then the proteins are digested into peptides, and finally, the peptides are injected into a mass spectrometer for analysis. The output from a mass spectrometer is a set of MS/MS spectra, one for each detected peptide variant. The bottom half illustrates the major steps of a computational analysis to infer protein identities from MS/MS spectra. The first step is to interpret the spectra to produce peptide and post-translational modification (PTM) predictions, then followed by refinement of the PTM predictions. Based on the peptide and PTM predictions, the last step is to infer the protein composition in the sample.
from various steps of an MS experiment can accumulate. These steps include the experimental protocol, experimental instruments and subsequent computational analyses. In particular, problems in the area of experimental protocols and mass spectrometers, such as non-ionizable peptides, incomplete protein digestion, undersampling where low abundance proteins are often missed and noise signal in the spectra, are well documented [3]. Advancements in experimental protocols, mass spectrometers and software are underway in the proteomics community to address these issues.

There are several major challenges for MS/MS computational analyses. They include identification of peptides from experimental spectra, especially those with post-translational modifications (PTMs), as well as the inference of protein composition based on the putative identified peptides. PTMs are naturally occurring chemical modifications of proteins that are known to play vital roles in cellular processes, such as protein regulation. In an MS/MS experiment, they are characterized by their delta mass (a change in the peptide mass) and their preferred target residues (amino acids that are modified). For example, the PTM phosphorylation has a mass shift of \( \sim 80 \) Da and targets only serine (S), threonine (T) and tyrosine (Y) residues. Therefore, a predicted peptide sequence with a delta mass of \( \sim 80 \) Da and the modification estimated to be on one of its S, T or Y residues can be interpreted as a PTM prediction for a phosphorylated peptide (phosphopeptide). To distinguish between predictions of unmodified and modified peptides, we will define those mapping to unmodified peptides as peptide predictions and those referring to modified peptides as PTM predictions.

The first challenge of an MS/MS computational analysis is peptide prediction, which decipher information embedded in spectra to produce a set of likely peptide predictions for both unmodified peptides and those with PTMs [5, 6, 7, 8]. Despite numerous research advancements, for complex biological samples, current state-of-the-art spectral analysis algorithms can only successfully map a small fraction (< 20%) of spectra to real peptides [9, 10, 11]. The presence of PTMs further complicates this matter since a
significant percentage of modified peptide predictions has false PTM assignments [12].

Refinement of PTMs predicted from MS/MS is an ongoing research area.

The second major challenge of an MS/MS computational analysis is protein prediction after peptide predictions are made. To identify proteins that generated the noisy spectra, false peptide predictions need to be filtered out first, and then the remaining predictions need to be assigned to their most likely corresponding proteins. A simple peptide score threshold for removing false peptide predictions was shown to be inadequate in a number of studies [10], [11], [13], [14]. In addition, ambiguous peptide-to-protein mappings add to the difficult task of protein identification. Methods that use heuristic and probabilistic approaches have been proposed to account for these issues [13], [15]. The inclusion of prior knowledge about proteins, such as protein function annotation and gene expression data, has helped to increase protein coverage and reduce false protein identifications in the two studies [10], [11]. However, even for the benchmark dataset used in the two aforementioned studies [10], [11], a large portion of proteins expected to be in the sample cannot be detected. Limited protein coverage, especially for low abundance proteins, is a general problem with high-throughput protein profiling with MS.

In this thesis, we set out to tackle the following challenges in an MS/MS analysis: 1) the refinement of PTM predictions from experimental spectra and 2) the inference of protein composition based on peptide predictions. These correspond to the last two major computational MS spectral analysis steps, highlighted in Fig. [11]. We make use of existing spectral analysis algorithms to interpret spectra and use their peptide predictions as input to our algorithms. Individually, we show that our two PTM refinement algorithms outperform state-of-the-art algorithms and our protein identification algorithm performs favourably to the current methods. Collectively, in an end-to-end MS/MS computational analysis of a human chromatin protein complex study, we demonstrate that our analysis pipeline can find putative protein complex members. Moreover, it can provide valuable insights into the formation and regulation of protein complexes by detailing the specificity
of different PTMs for the members in each complex.

1.1 Organization of the Thesis

The remaining chapters of the thesis are organized as follows. Chapters 2 and 3 provide a background on the current state of MS/MS analysis and computational concepts used in this thesis, respectively. In chapters 4 and 5 we tackle the problem of PTM refinement. Chapter 4 describes a novel PTM refinement algorithm PTMClust and outlines our generative modelling approach to the problem, which was published in Bioinformatics [12]. Improving upon PTMClust, chapter 5 presents an improved method for solving the PTM refinement problem, which is based on nonparametric Bayesian modelling and utilizes a Markov chain Monte Carlo (MCMC) inference algorithm. Chapter 6 focuses on the problem of protein identification from MS/MS experiments and describes a novel approach using hierarchical clustering, which leverages on our new clustering algorithm, Hierarchical Affinity Propagation, published at the Conference on Uncertainty in Artificial Intelligence [16]. Chapter 7 concludes by summarizing our work and future research directions.
Chapter 2

Background on Proteomics Analysis

Proteins are macromolecules that are essential parts of a living organism and facilitate a number of processes within the cell. Proteins provide a wide range of important functions, such as catalysis of biochemical reactions, formation of scaffolds that maintains cell shape, cell signalling, immune responses, cell adhesion, nutrient storage, and transportation of molecules (including other proteins) between subcellular organelles and across the cell membrane [17]. Proteins are made up of a chain of amino acids with various lengths, ranging from tens to thousands of subunits (residues) long.

Protein biosynthesis is a complex process. Genes (segments in the DNA that encode functional molecules) are converted into proteins, which are biological macromolecules, through the processes of transcription and translation. First, transcription involves creating an intermediate molecule called ribonucleic acid (RNA) from DNA. Then these RNAs, specifically a special type of RNAs called messenger RNA molecules (mRNAs), have unwanted segments spliced before they are translated into proteins. Fig. 2.1 illustrates the steps in the protein biosynthesis process. Both transcription and translation, along with DNA replication, are the basis of the Central Dogma of Molecular Biology [18], which outlines the flow of genetic information in the cell.

At the time of translation, a protein can either be active or inactive, and its activity is
Figure 2.1: Protein biosynthesis process with post-translational modification (PTM). Although it is depicted as post-translation, a PTM can occur during or after a protein is translated from an mRNA.
generally regulated by a chemical modification. This protein modification, generally referred to as a post-translational modification (PTM), can be either co- or post-translation and involves the addition of a chemical group, a mutation of an amino acid from one to another, or the removal of amino acids from the beginning of the protein \[17\]. Modification sites, positions where PTMs occur in proteins, are generally site-specific, which means they do not occur randomly and are found at a specific set of amino acids for each PTM. A modification site is believed to be correlated with the type of PTM, the properties of the amino acids, proximity of functional protein sites, and the three-dimensional configuration of its location. A modification to a protein usually results in a change to its three-dimensional shape, function and interaction with other molecules. It has been shown that the presence of a PTM can regulate a protein’s function by either activating or suppressing it \[17\].

Phosphorylation is a good example for illustrating the importance of PTMs in the cell and the complex mechanism that occurs during modification. Phosphorylation is one of the most well-studied PTMs because it regulates the activity of many enzymes (proteins that catalyze biochemical reactions) and receptors (proteins that initiate cellular responses by binding to specific ligands, which are small molecules such as neurotransmitter and hormones) \[17\]. Phosphorylation and dephosphorylation are catalyzed by specific enzymatic protein families called kinases and phosphatases, respectively. In phosphorylation, a protein kinase transfers a phosphate group from a high-energy donor molecule such as adenosine 5’-triphosphate (ATP) (a special molecule used throughout the cell as energy storage) to the protein being phosphorylated (substrate). This can serve to activate the substrate by inducing a conformational change in the structure of the protein. Similarly, dephosphorylation (removal of the phosphate group) by phosphatases can serve as a mechanism to inactivate the phosphorylated protein by removing the phosphate group through hydrolysis.

Since proteins play a significant role in cell functions, a disruption in normal protein
function can lead to a catastrophic effect in the cell and, consequently to the host organism. Similarly, since PTMs are a major part of the biological mechanism regulating protein function, a disruption in PTM-related processes can also have a catastrophic effect on the host organism. Diseases like Parkinson’s and Alzheimer’s are great examples of such ill effects. Mutation to the protein parkin, a key component in the ubiquitin-mediated proteolytic pathway (ubiquitination is another known PTM), has been shown to lead to the death of nigral neurons, a leading cause of Parkinson’s disease [19]. Likewise, abnormal hyperphosphorylation of the microtubule-associated protein tau is found to occur in patients with Alzheimer’s disease [20], another neurodegenerative disease. Moreover, mutations of genes at or near the insulin signalling region, which can be PTM impairing, have been shown to be a major factor in Type 2 diabetes mellitus [21].

The ability to accurately profile protein expression, and recently PTM expression, on a genome-wide scale is a fundamental problem in qualitative proteomics and disease-related studies like biomarker discovery. In this chapter, we first present a prominent experimental method for high-throughput protein and PTM profiling based on MS. We then conclude with a discussion of the major areas in a computational MS spectral analysis pipeline: mapping spectra to peptides (peptide predictions), prediction of peptides with PTMs, PTM refinement, PTM localization scoring and mapping peptides to proteins (protein predictions).

2.1 Bottom-up Analysis of Protein Mixture using Tandem Mass Spectrometry

With recent technological advancements, MS has emerged as the preferred method for high-throughput proteomics characterization such as protein profiling [3, 22] and quantification [23, 24]. This technique allows for the identification of thousands of proteins simultaneously from complex mixtures such as human tissue extracts. One of the first ap-
Chapter 2. Background on Proteomics Analysis

Figure 2.2: Components of a tandem ion trap mass spectrometer. The first MS stage measures the mass of the incoming molecules (e.g., peptides). Next, molecules with the same mass are grouped, isolated and broken down into ion fragments. In the MS/MS stage, the mass spectrometer captures and outputs the ion fragmentation pattern for each group of molecules as an MS/MS spectrum. *The figure is reproduced from [34].

Figure 2.2: Components of a tandem ion trap mass spectrometer. The first MS stage measures the mass of the incoming molecules (e.g., peptides). Next, molecules with the same mass are grouped, isolated and broken down into ion fragments. In the MS/MS stage, the mass spectrometer captures and outputs the ion fragmentation pattern for each group of molecules as an MS/MS spectrum. *The figure is reproduced from [34].

proaches using MS to perform high-throughput studies of PTMs was proposed by Yates et al. [25] in 1995. Since then, a number of studies using MS has been successful in identifying PTMs involving hundreds of sites of phosphorylation, ubiquitination, oxidation, methylation as well as acetylation in yeast, and mouse and human tissues [26, 27, 28, 29]. In-depth reviews of protein MS, detection of PTM by MS and computational tools designed for MS analysis can be found in [30, 31, 32, 33].

At the core of an MS-based experiments is an analytical instrument called a mass spectrometer that measures masses and relative concentrations of atoms and molecules. A mass spectrometer consists of three basic components: an ion source, a mass analyzer, and a detector. Shown in Fig. 2.2 is an ion trap mass spectrometer. The spectrometer uses an ion source which is an electrospray ionizer, and the mass analyzer is made up of a vacuum chamber (use to collect the ion particles) and a collision cell (use to break the ions into pieces for further analysis). Finally, the mass (more specifically the mass-to-charge ratio (m/z)) is measured from the dynamics of the charged particles in the electric and magnetic fields inside the vacuum chamber. The mass can be calculated
by applying both the Lorentz force law \( F = q(E + v \times B) \) and Newton’s second law of motion \( F = ma \). Rearranging the equations, we obtain the formula for calculating mass as

\[
\frac{m}{q} = \frac{E + v \times B}{a},
\]

where \( F \) is the force applied to the ion, \( m \) is the mass of the ion, \( a \) is the acceleration, \( q \) is the ionic charge, \( E \) is the electric field, and \( v \times B \) is the vector cross product of the ion velocity and the magnetic field.

An MS experiment requires proteins of interest to be presented in a relatively pure form and broken down into short peptides using an enzyme such as trypsin. Although analysis of whole protein is possible, we will focus only on MS studies where proteins are digested by trypsin into short peptides and typically have lengths between 3 to 36 amino acids long. Methods using MS to analyze whole proteins are far less popular and are still in early stage of development. After the peptide mixture is ionized by electrospray ionization and injected into the mass spectrometer, the machine scans and identifies the mass of each molecule in the sample by measuring \( v \) and \( a \) and applying Eq. 2.1. For a complex mixture, it is common to simplify the mixture by separating peptides based on their chemical properties, such as hydrophobicity using liquid chromatography, before it is ionized and injected into a mass spectrometer. This separates the mixture into smaller samples and allows the mass spectrometer to identify a larger fraction of the peptides. The speed at which a molecule is scanned and its mass is measured affects the number of peptides that can be identified. In an MS experiment profiling a complex mixture of mouse tissues, for example, seven technical replicates of the same sample were needed to reach saturation in protein coverage [3].

Once in the mass spectrometer, peptides are grouped and isolated based on their mass. Ideally, each group will contain one peptide variant. For each group the peptides are further broken down by a fragmentation method, such as collision-induced dissociation (CID), to produce ion fragments. The mass spectrometer captures the ion fragmentation
pattern for each group in a mass spectrum. A spectrum is a series of ion signals, where each ion signal consists of a mass (m/z) and intensity; each ion signal is called a peak. A theoretical spectrum is shown in Fig. 2.5.

Before continuing, we outline two assumptions that will simplify our discussion onwards. First, we assume spectra are generated for peptides that are fragmented by CID. Different fragmentation methods produce spectra with different types of ions, but the general concept behind their analysis is the same. Second, we assume the ions produced are singly charged. Since mass are measured as mass over charge (m/z), multiple charged ions produce observed mass that are proportional to their singly charged counterpart; they can be easily accounted for. Conversely, uncharged ions cannot be detected by mass spectrometers and will result in missing peaks. How different spectral analysis algorithms handle this problem is discussed below.

Instrument parameters are usually chosen such that fragmentation occurs primarily along the peptide amide bond (also known as peptide bond) backbone to generate informative b- and y-ions (see Fig. 2.3), where each fragmented ion corresponds to a peak in the spectrum. This means that each fragmented peptide can produce one b-ion and a corresponding y-ion; the masses of the pair of ions sum to the mass of the peptide. This is demonstrated in the ladder of ion pairs of our example peptide in Fig. 2.4. Furthermore, the observed mass difference between an adjacent pair of peaks (assuming no noise) corresponds to the mass of the specific intervening amino acid residue [35]. In an ideal case, each unique peptide sequence produces a distinctive spectrum that acts as its signature. However, in practice spectra are difficult to interpret because some of the ions are not detected due to reasons such as incomplete fragmentation and uncharged ions, and noise peaks due to contamination and machine noise are found frequently. Moreover, the problem worsens with the presence of PTMs, which shifts the mass of the corresponding ions and changes the ion fragmentation pattern significantly.

Key computational analyses start after the acquisition of experimental spectra. Sim-
Figure 2.3: An overview of the structure of a peptide, and the b- and y-ion that are formed by fragmentation along the peptide amine bond. (a) shows an overview of the structure of a peptide of two neighbouring amino acids connected by a peptide amine bond (peptide bond). The side-chain specific to each amino acid is represented by $R$. (b) displays the b- and y-ion that are formed as a result of breakage at the amine bond. The CID fragmentation method is designed to fragment peptides at one of their peptide amine bonds to produce corresponding b- and y-ions. The ion fragment that contains the N-terminus of the peptide is called the b-ion. Similar, the ion fragment that contains the C-terminus of the peptide is called the y-ion.
Figure 2.4: Example of a mass fragmentation ladder. It shows the set of hypothetically possible fragmented ion pairs (b- and y-ions) for the peptide SPAFDSIMAETLK that can occur during tandem mass spectrometry (MS/MS). Each b-ion has a corresponding y-ion that when their masses are combined equal to the mass of the pre-fragmented peptide. Ideally, each fragmentation event occurs at the peptide amide bond backbone causing breakage at one of the bonds connecting adjacent amino acids. *The figure is reproduced from [34].
ilar to an assembly line, each step depends on results from previous steps. As such, breakdown at any one step can negatively impact the final protein predictions.

2.2 Mapping Spectra to Peptides

The problem of peptide prediction based on spectra is an active research area however is far from solved. A number of studies analyzing complex biological samples revealed <20% of spectra can be mapped to real peptides with state-of-the-art approaches [9, 10, 11]. Current spectral analysis algorithms can be classified into four main categories: database search, de novo sequencing, sequence tag (hybrid) and spectra matching. Although not discussed in detail, an important preprocessing step generally included in these algorithms is low-quality spectra filtering, which removes noisy spectra to reduce false predictions and reduces computational time. An example of a low quality spectra filter is simply removing spectra with maximum peak intensity less than a predefined threshold. We now discuss all of the aforementioned methods.

2.2.1 Database Search Algorithms

The class of database search algorithms leverages on two key properties of spectra: the combined mass of each b- and y-ion pair is equal to the precursor mass (mass of the pre-fragmented peptide) and the mass difference between an adjacent pair of peaks corresponds to the mass of the specific intervening amino acid residue [35]. An example of a peptide and its resulting ions are given in Fig. 2.4. Based on these properties, the sequence of the underlying peptide can be deciphered by successively interpreting each peak pair. An overview of the basic approach of a database search algorithm is outlined in Fig. 2.5, where it assumes an ideal case. However, in practice, many ions are undetected and spectra contain numerous noise peaks, which cause interpretation of the spectra difficult. Various methods have been developed to counter these problems.
Figure 2.5: Theoretical example of a peptide prediction using the basic approach of a database search algorithm on a noiseless spectrum containing only b-ions. Using a peptide list as a guide, the algorithm predicts the underlying amino acid sequence for this spectrum. As the algorithm progresses through the spectrum, any peptides that cannot match the sequenced amino acids are removed from the reference list. Each peak represents a fragmented ion, and the mass interval between two neighbouring peaks corresponds to an amino acid. In (a) the algorithm starts with the first peak interval (i.e., from the start to the first peak) and determine the amino acid that corresponds to that mass, which in this case is A. After each amino acid match, the algorithm removes any peptides in the reference peptide list that cannot match to the spectrum thus far. The algorithm repeats this procedure (b), skipping any unexplainable peak intervals, until all peak intervals are examined (c). In the end, the remaining peptides in the reference peptide list can be used to fill in the gaps in the peptide sequencing if gaps exist.
Popular database search methods used to analyze peptide spectra include Sequest [5], Mascot [6], OMSSA [36], X!Tandem [7] and SQID [37]. A database search algorithm first extracts peptide sequence candidates generated *in silico* from a reference protein database using the precursor mass (mass of the pre-fragmented peptide) and input cleavage specificity (e.g., the enzyme trypsin cleaves proteins after each lysine (k) and arginine (R) to produce peptides) as filtering criteria. It then scores each candidate against the experimental spectrum based on the set of matching peaks’ mass and intensity. The candidate spectra are restricted to only those with total peptide mass within a tolerated distance to the precursor mass. By using a reference database of proteins, interpretation of spectra with missing ion peaks is possible. Restriction of candidate peptides to only those that are generated from the database greatly reduces the number of peptide candidates, which helps to reduce false positive predictions and improve the computational time.

Different algorithms employ diverse methods for matching experimental and theoretical spectra, and for scoring the matches. The Sequest method continues to be a very popular database search algorithm despite numerous recently developed options. During the spectra matching stage, Sequest matches only the top \( n \) peaks with the highest intensity, where \( n \) is fixed (since the source code is not publicly open, detail of the algorithm is not available and the value of \( n \) is unknown). These peaks are normalized to uniform intensity during comparison between each experimental and theoretical spectra pair so that peaks’ intensities are not considered since they are difficult to model. A set of candidate peptide sequences are selected based on their preliminary scores, which are sums of peak scores for the selected peaks. Peak scores are calculated based on the existence of matching b- or y-ions, nearby isotope peaks (\( \pm 1, 2 \) or 3 Da from the original peak) and intensity of the original ion. A cross-correlation score (\( X_{corr} \)), calculated using Fast Fourier Transform, is assigned to each peptide-spectrum mapping (PSM) (same as peptide prediction). The algorithm produces multiple PSMs per spectrum but only the top hit according to \( X_{corr} \) is generally used. It has been shown that \( X_{corr} \) alone is
not enough to distinguish between true and false predictions; hence, a number of PSM scoring methods (discussed in Sec. 2.2.5) have been developed to evaluate Sequest results.

Database search algorithms are dependent on having a reference protein database and is severely limited by the completeness and accuracy of the database. They cannot identify proteins not present in the database, account for sequencing errors and mutations (e.g., single point mutations), or be used for organisms with unknown genome sequences. Furthermore, there are several issues that plague the performance of database search algorithms. These include the presence of ions other than b- and y-ions that can be generated by CID, noise in the spectrum due to contamination and background noise, incomplete peptide fragmentation, multiple amino acids having the same or very similar masses (e.g., leucine and isoleucine), various species of peptides with the same mass isolated and fragmented together in MS/MS scans, naturally-occurring post-translational modifications (PTMs) and artificial modifications (introduced during experimental procedure) of proteins. Of these, both natural and artificial modifications can be accounted for by explicitly searching for them [25, 38] or blindly blanketing all possible modifications with a specialized PTM search algorithm, such as SIMS [8], InsPecT [39], GutenTag [40], PILOT-PTM [41] and TagRecon [42]. The inclusion of PTM searches may result in a higher false detection rate and takes a significantly longer time to complete.

Lastly, the choice of the database are major confounding factors. For instance, correct peptides might be missing if the database is too small. However, as the size of the database increases, the chance of matching the experimental spectrum randomly to an incorrect peptide sequence and the computation time required to complete the analysis significantly increase. To estimate the false detection rate, in addition to real proteins, decoy proteins, generated by reversing the sequence of real proteins, are often used [9, 22, 43, 44, 45, 46, 47, 48]. Here, peptide predictions matching to peptides taken from these decoy proteins are considered false positives. Using the detection of real and decoy peptides as a quality indicator, for genome scale studies, it has been reported that < 20%
of the spectra can be confidently mapped to real peptides [9, 10, 11]. This is an issue that will affect further downstream analyses such as protein identification.

### 2.2.2 De Novo Sequencing Algorithms

*De novo* sequencing algorithms follow the general concept of interpreting a spectrum by stringing together amino acids corresponding to the mass difference between adjacent peaks. They differ from database search algorithms by having peptides reconstructed solely from the spectrum without recourse to a reference protein database. This property allows *de novo* algorithms to identify novel proteins and, more importantly, be used for analyzing unsequenced organisms. For the example given in Fig. 2.5 for database search algorithms, imagine following the steps outlined without the guidance of a list of reference proteins. In this ideal case, both *de novo* sequencing and database search algorithms work well.

A number of *de novo* sequencing algorithms have been developed, including Lutefisk [49, 50], Sherenga [51], PepNovo [52], PEAKS [53], EigenMS [54], NovoHMM [55], AUDENS [56], MSNovo [57], PILOT [58] and others [59, 60, 61]. A common theme shared by many of these algorithms is the use of a spectrum graph [62], which is a graph representation of a spectrum. In a spectrum graph, each vertex represents a peak in the spectrum, and an edge between vertices is drawn only if their mass difference is equal to mass of one of the amino acids. Methods such as dynamic programming [53, 56, 59, 60], probabilistic likelihood approach [51, 52] and other non-probabilistic methods [49, 50, 54] have been used to interpret spectrum graphs. Other methods alternative to the spectrum graph include Hidden Markov Model [55], integer linear optimization approach [58] and a mass array-based dynamic programming approach [57].

Similar to database search methods, *de novo* sequencing algorithms are adversely affected by spectral noise. This results in numerous spurious peak pairings. This may lead to erroneous results if these false pairings coincidentally have mass difference equal to an
amino acid mass. Additionally, complete fragmentation is rarely observed, which causes gaps in amino acid sequence interpretations. Therefore, it is uncommon for de novo algorithms to produce full-length peptide sequences for most spectra; generally, confident results have multiple short sequences of amino acids (sequence tags) per spectrum. Many algorithms are efficient in finding sequence tags of length 3-6. Full-length peptide sequencing by de novo might be an elusive goal due to the relatively poor quality of spectra and the computational time it takes to generate longer sequences. In addition to sequence tags, some algorithms provide gapped peptide interpretations, which are continuous strings of sequence tags with gaps marked by the value of missing mass at the location of the gaps [63]. Alone, de novo sequencing algorithms are seldom used to analyze spectra. However, combined with a database search algorithm it forms a class of spectral analysis methods called sequence tag algorithms that is very popular.

2.2.3 Sequence Tag Algorithms

Sequence tag (hybrid) algorithms combine the strengths of database search and de novo sequencing algorithms. These methods generally consists of two steps: first, analysis of spectra using a de novo sequencing approach that generates sequence tags (length 3 for most algorithms) per spectrum, and second, inference of full-length peptides, using the former tags as anchors, with a database search method. Benefits of this type of approaches over database search methods are twofold. First, the sequence tags generated can contain PTMs, which allows for an efficient and unrestricted PTM search that many database search algorithms are not designed for. Second, computation time and false prediction rate are reduced because sequence tags, serving as filtering criteria, limit the potential candidate peptides that are considered during the database search stage. However, similar to the database search methods, these methods are highly affected by the completeness and correctness of the reference protein database, and are sensitive to the quality of the spectra. Their advantages over de novo methods are their abilities to
simultaneously fill in gaps between sequence tags and discard false sequence tags that are not part of the final peptide predictions; full-length peptide predictions are produced with this method whereas with \textit{de novo} it is generally impossible in practice.

Two of the fastest sequence tag algorithms, InsPecT \cite{39} and Paragon \cite{64}, claimed to have orders of magnitude speed improvement over Sequest. A recently published algorithm, DirecTag \cite{65}, make use of statistical models to infer sequence tags. When compared to InsPecT, DirecTag is shown to markedly outperform it. Similar to the sequence tag approach, Kim \textit{et al.} \cite{66} proposed an algorithm that creates a spectral dictionary and then applies a database search algorithm using the spectral dictionary in place of a reference protein database. A spectral dictionary is a set of generated spectra of all possible full-length peptides from a reference database using a \textit{de novo} sequencing method. The sequence tag methods do have drawbacks. They are limited to only peptides of charge +2 (peptides with other charge states are difficult to model) and are error-prone to finding long peptide sequences (length > 10).

\subsection{Spectra Matching Algorithms}

Amongst the four types of spectral analysis algorithms, the spectra matching methods are potentially the fastest. They trade the ability to analyze unseen spectra for fast interpretation. Leveraging the enormous and growing collection of interpreted experimental spectra, these methods match the experimental spectra against referenced, previously interpreted spectra to produce peptide predictions. The core principle of these methods is that multiple experimentally generated spectra for the same underlying peptide are more likely to match to each other than to a theoretically generated spectrum. This is because peak intensity, peak mass variation, presence or absence of peaks, isotope peaks distribution and background noise are difficult to model accurately. A glaring limitation is that they cannot find any new proteins (and peptides) that have not been previously reported. In addition, they work only if both experimental and referenced spectra
are produced under similar experimental conditions, which include comparable sample preparation protocols, fractionation procedures and the type of mass spectrometer. As an example of a conflict, the peptide fragmentation techniques CID and electron-transfer dissociation (ETD) produce very different types of fragmented ions, which lead to spectra with unique fragmentation patterns. Furthermore, the protein coverage of reference spectra and their prediction quality affect the methods’ applicability and accuracy.

A major research focus in spectra matching algorithms is on developing an accurate evaluation metric for comparing two spectra that is also computationally efficient. A prominent method used in many spectral analysis algorithms is a dot product metric to measure the degree of similarity between experimental and referenced spectra. Algorithms that employ this method includes SpectralST [67], X!Hunter [68], MSPepSearch [69], BiblioSpec [70] and the method proposed by Frewen et al. [71]. A recently published algorithm, Pepitome [72], employs an alternative scoring method that relies on a probabilistic scoring metric that is designed to address two issues with the dot product metric: lack of statistical interpretation and fragment ion mass (m/z) discrepancies are ignored in the scoring. The result shows an improvement over SpectralST when identifying peptides from complex MS/MS datasets. The performance improvement potential over other classes of spectral analysis algorithms makes spectra matching algorithms appealing in analysis of large-scale studies if applicable.

2.2.5 Scoring Peptide Predictions

The above discussed spectral analysis algorithms will provide a PSM (peptide-spectrum mapping, also referred as peptide prediction) for each analyzed mass spectrum regardless of the quality of the result. The difficult task lies in separating true from spurious PSMs. Many current PSM evaluation algorithms focus on providing a score for each PSM, in the form of a confidence score, a \( p \)-value or a \( z \)-score [9, 22, 43, 44, 45, 46, 47, 48]. In the following, we provide a description of PeptideProphet, a popular method for
scoring Sequest results. To facilitate the calculation, it is common to analyze spectra against a decoy protein database (for database search and hybrid algorithms) \cite{9, 22, 43, 44, 45} or reference spectra containing PSMs to decoy proteins (for spectra matching algorithms) \cite{13}, and estimate a false discovery rate based on the number of hits to decoy proteins. Other approaches include ranking PSMs \cite{73, 74} and combining scores from multiple algorithms \cite{75}.

**PeptideProphet**

PeptideProphet \cite{9} is a statistical model that provides a probability score for estimating the accuracy of PSMs. For each PSM, it takes in, from a database search algorithm, a number of scores and features, and converts them into a discriminant score $F$. For Sequest, it uses four scores: the cross-correlation score $X_{corr}$ (described above) corrected for peptide length, the difference between the top score to the second best score ($\Delta Cn$), a measure of how well the assigned peptide scored relative to those of similar mass in the database ($SpRank$), and the mass difference between the mass of precursor ion and the assigned peptide ($d_M$). The discriminant score is a weighted combination of the above scores with weights learned using training data. As an example, PeptideProphet uses the following discriminant score for +2 charge human peptides:

$$F = 8.4 \times \frac{\ln(X_{corr})}{\ln(N)} + 7.4 \times \Delta Cn - 0.2 \times \ln(SpRank) - 0.3 \times d_M - 0.96,$$

(2.2)

where $N$ is the length of the assigned peptide.

Given a distribution of $F$ for all PSMs, the algorithm employs the EM algorithm (Sec. 3.2.5) to fit a mixture of two Gaussians model (Sec. 3.2.6) that captures the distribution of $F$ for correct and incorrect PSMs. The probability score for each PSM is calculated from the final model.

PeptideProphet provides an elegant and simple way to compute the probability score for each peptide prediction. However, the learned model is specific to a spectral analysis
algorithm, the type of mass spectrometer and the organism the sample originated from. The model needs to be retrained for different experimental settings as outlined in [9].

2.3 Predicting Peptides with PTMs

PTMs can be found using a database search algorithm by allowing for offsets in the peaks, with the sum of the offsets being the modification mass and the location of the offsets as the modification sites [4]. Shown in Fig. 2.6 is an example spectrum with a PTM that illustrates how database search algorithms can be used to predict the peptide sequence. Similar to non-PTM spectra, a database search algorithm takes steps to sequence the peptide until it reaches a peak interval which cannot be explained by any known amino acid. Here, the algorithm skips to the next interval, while also skipping to the following amino acid in the reference peptides, and continues until sequencing is completed. When a confident prediction is identified, the unsequenced interval can be inferred from the predicted peptide sequence with the modification mass equal to the mass difference between the unsequenced interval and its associated residue.

Two of the most widely used database search algorithms are Sequest [5] and MASCOT [6]. Although originally designed to predict normal unmodified peptide sequences from mass spectra, they can be used, with slight modifications to their algorithms, to predict PTMs [26, 27, 28, 29, 76]. The resulting restrictive approach requires a predefined list of PTMs (modification mass and possible modified amino acids for each PTM) as input. In addition to regular searches, these algorithms enumerate through and score each of the predefined modifications. This exhaustive search approach has serious limitations since it is only feasible when considering a small number of PTMs and cannot search for any unknown modifications because the list of PTMs must be predetermined.

A new class of PTM search algorithms has emerged that can perform blind (unrestrictive) PTM searches. In contrast to the restrictive approach described above, blind
Figure 2.6: Theoretical example of a peptide with a PTM predicted using a database search algorithm on a noiseless spectrum containing only b-ions. (a)-(d) show the basic steps that a PTM database search algorithm takes to predict the amino acid sequence for the underlying peptide. The algorithm proceeds similar to a database search algorithm, illustrated in Fig. 2.5. When a PTM occurs, the mass interval corresponding to the amino acid with the PTM has a shift in mass (c), which the algorithm cannot interpret initially. When this occurs, the algorithm skips this interval and continues. At the end, using the peptide(s) from the reference list that best match the mass spectrum as a guide, the uninterpreted interval can be sequenced (d) by considering the possibility of a PTM. In (e), it shows that the position of the modification site can become ambiguous when peaks are missing, which occurs frequently.
search algorithms require no predetermined list of PTMs and are able to consider all modifications at once, both previously known and new ones. In this thesis, we have chosen to base our study on two new blind PTM search algorithms, SIMS (Sequential Interval Motif Search) \[8\] and InsPecT \[39\]. Based on the Sequest database search algorithm, SIMS was developed to perform both normal peptide sequencing and blind PTM searches on MS spectra. To reduce the enormous search space associated with the naïve PTM search, SIMS uses heuristics to isolate a small subset of possible peptide candidates and modification masses before it correlates the experimental spectrum to the theoretical spectrum generated from the reference peptide. A more in-depth description of the SIMS algorithm can be found in Appendix E.

The InsPecT method uses the sequence tags approach (Sec. 2.2.3) to generate both unmodified and modified PSMs. This algorithm contains five main steps. For each spectrum, first, it produces sequence tags using a customized \textit{de novo} sequencing algorithm that works in the presence of PTMs. Second, it filters the reference database using a trie-based search method\[2\]. Third, it uses a dynamic programming technique to identify candidate peptides with modifications without explicit enumeration of peptides. Next, it applies its scoring algorithm to score and rank each candidate peptide according to their relative likelihood of generating the spectrum. Finally, using a set of training samples consisting of real and decoy proteins, a discriminative model is trained on the score and rank of each candidate peptide along with the number and total intensity explained by the candidate peptide, number of b- and y-ions found and delta score between the best and next best match. Based on this discriminating model, each candidate peptide is given a \(p\)-value score.

Several other blind search algorithms have also been proposed, and they employ

---

\[2\]A trie (also referred to as prefix tree) is a ordered tree data structure that is used to store map (or dictionary) abstract data types \[77\]. A trie has the peculiar feature that it takes the same amount of time to insert, delete or find an entry, which makes it appealing over other data structures like binary search trees and hash tables in applications that are expected to perform equal number of insert, delete and find operations.
various optimization techniques and sequence prediction approaches [78, 79, 80, 81, 82, 83, 84, 42]. These include dynamic programming, point process models, database filtration based on peptide sequence tags, and a combinatorics approach using both de novo and database search sequencing techniques. While some of the algorithms place more emphasis on PTM coverage and others on speed improvements, their overall results are comparable. When compared to other blind search algorithms, SIMS and InsPecT have been shown to have equal or better coverage and overall performance [8, 39]. Despite their general efficacy, only a small fraction of the total acquired spectra is typically deduced with high confidence by SIMS and InsPecT, or indeed, any other existing search algorithm. Furthermore, there is a lack of overlap between results from various algorithms, e.g., SIMS and InsPecT shown in [8]. The problems of spectra coverage and a lack of overlap between different algorithms are known issues with spectral analysis algorithms [9, 10, 11, 84]. The inclusion of PTMs has only complicated these issues [12]. We have attempted to address these problems in our study (Ch. 4 and Chap. 5) and show that post-processing with a PTM refinement algorithm can improve on the quality of PTM predictions, and the overall overlap between different blind PTM search algorithms.

2.4 Refinement of PTM Predictions

In practice, blind PTM search methods suffer from two major sources of error: 1) sequence-dependent uncertainty in the modification position (residue position along the peptide sequence where the modification is deemed to occur) and 2) mass inaccuracy for the modification mass. The fragmentation process is often incomplete, and the presence of labile PTMs may interfere with this process [85]. Both issues combined result in spectra missing peaks that in turn may lead to ambiguous or erroneous modification predictions. Fig. 2.6(e) illustrates how an incomplete fragmented spectrum can affect the determination of the modification site. The presences of natural stable isotopes, such
as carbon-13, in addition to electronic noise are major contributors to inaccurate mass measurements. These issues are more prominent in spectra generated from low mass resolution mass spectrometers (e.g., ion trap mass spectrometers), which are still commonly used in today’s MS studies.

These two sources of error were acknowledged by Tsur et al. [78], who briefly described a heuristic approach to account for ‘shadows’ (modifications that are misplaced by a PTM search engine), and later by the same group in the PTM refinement algorithm PTMFinder [86]. However, in the former method, their approach favours high abundance modified peptides, since it requires each peptide match to occur multiple times; discretizes observed modification masses, which introduces additional error with the mass measurements; and can handle only one type of error per peptide (namely either a modification mass error of exactly 1 Da or a modification position misplaced by exactly one residue). The latter method, PTMFinder, takes a machine learning approach where it groups and reanalyzes spectra mapping to the same modified peptide sequence to produce for each spectrum a final peptide sequence with a modification mass and a modification position. This method also suffers from favouring high abundance modified peptides and discretizing observed modification masses. As we show later in Ch. 4, only a small fraction of observed modified peptides is identified multiple times in a typical genome-wide MS study. These restrictions limit the suitability of both methods’ error correction approach to global PTM studies. We note that in addition to correcting for errors with modification mass and modification position estimations, PTMFinder can refine the peptide sequence and provides a $p$-value confidence score for both the reported peptide sequence and the modification.

The errors associated with blind PTM search algorithms could be addressed either by developing an algorithm that can denoise the errors associated with measuring masses and identifying modification sites (PTM refinement algorithm) or by technological improvement in the mass spectrometer (such as higher mass accuracy). Although high mass
accuracy mass spectrometers can theoretically reduce modification mass errors, they are not known to eliminate the need for modification site correction. Therefore, data generated from high mass accuracy mass spectrometers can still benefit from the use of PTM refinement to obtain accurate PTM predictions.

### 2.5 Scoring Predicted PTM Sites

Instead of refining observed modification masses and modification positions, PTM localization scoring methods provide a way to evaluate the quality of predicted modification sites from PTM search engines. The two main strategies for scoring the reliability of modification site localizations are: 1) to calculate the probability that a peak responsible for the site determination is matched at random and 2) to compute the search engine score difference between predictions with varying site localizations. Methods that use the former strategy include A-score [76], PTM Score (embedded in MaxQuant and Andromeda) [87], the Phosphorylation Localization Score (PLS) in InsPecT [88], SLoMo [89], Phosphinator [90], PhosphoRS [91]. Examples of the latter scoring strategy are Mascot Delta Score [92], the SLIP score in Protein Prospector [93] and the variable modification localization (VML) score in Spectrum Mill [94]. A review of the different modification site scoring localization methods is provided in [95].

Similar to PTM refinement algorithms discussed above, a modification site localization scoring algorithm can be used to refine modification sites by taking the highest scoring position for each modified peptide. However, in addition to their underlying algorithm, there are three areas that modification site localization scoring methods differ from PTM refinement algorithms. First, a predefined list of PTMs is required for these scoring methods. Second, these scoring methods assume that input predicted modification masses are error-free and are mapped precisely to one of the PTMs in the predefined list. Lastly, most of these scoring methods are designed to score only phosphorylated
Consequently, modification site localization scoring methods are ill-suited to analyze PTM datasets generated from blind PTM search engines.

2.6 Mapping Peptides to Proteins

Despite advancement in peptide prediction and scoring techniques, inferring the underlying proteins remains a challenge [10, 11, 13]. The naïve method, when a reference database is used, is simply to take proteins that generated the candidate peptides in the set of high confidence PSMs [15]. However, many peptide sequences can map to multiple proteins because of naturally and artificially occurring phenomena such as isoforms due to alternative splicing, proteins from the same protein family, sequencing errors at either the DNA or protein level, and redundancy in the reference protein database, i.e., two different entries in the protein database for the same protein.

The latest development in protein prediction focuses on modelling the likelihood of proteins’ presence [10, 11, 13, 22, 96, 97]. The algorithm ProteinProphet (Sec. 2.6.1) models the probability of a protein’s presence by combining the likelihood of its supporting peptide predictions while taking into account how unique the peptides are within the reference database. On the other hand, the algorithm MSpresso (Sec. 2.6.2) combines protein prediction scores, taken from methods such as ProteinProphet, with gene expression data to recalculate prediction scores. Similarly, MSnet (Sec. 2.6.3) is another protein prediction re-scoring method that improves on an initial set of protein prediction scores by considering protein annotation taken from Gene Functional Network (GFN) [98, 99] as prior knowledge.

We now discuss in detail the three aforementioned protein prediction algorithms, ProteinProphet, MSpresso and MSnet.
2.6.1 ProteinProphet

Built on top of the peptide prediction scoring method PeptideProphet (Sec. 2.2.5), ProteinProphet [13] generates a minimal protein list sufficient to account for the input set of peptide predictions and produces probabilities that indicate whether proteins are present given the peptide predictions. This method favours proteins with multiple peptides assigned to them. Non-unique peptides, i.e., peptides with amino acid sequences common to several proteins, are assigned to all their corresponding proteins with their contributions weighted according to the number of proteins they are associated with. Furthermore, this method groups together proteins that share the exact set of peptide predictions.

The algorithm first calculates the probabilities of peptides being correct and incorrect, and then calculates the protein probabilities given these peptide probabilities. Given peptide predictions and their corresponding confidence scores, the model computes the probability that a peptide is correct given peptide prediction probabilities as

\[
p(+|D) = \frac{p(D|+)p(+)}{p(D|+)p(+) + p(D|-)p(-)},
\]

where + and − correct and incorrect predictions, respectively; \(D\) is the input values for peptide prediction \(i\), such as database search scores, number of tryptic termini and number of missed cleavages; \(p(D|+)\) and \(p(D|-)\) are the probabilities of an assigned peptide to a spectrum having information \(D\) amongst correctly and incorrectly assigned peptides, respectively; and \(p(+)\) and \(p(-)\) are prior probabilities that are set to the overall proportions of correct and incorrect peptide assignments in the dataset, respectively.

The model estimates the relative weight that a peptide \(i\) corresponds to a protein \(n\) in the set of all proteins \(N_s\) to adjust for peptides that map to multiple proteins, given as

\[
w_i^n = \frac{P_n}{\sum_{s=1}^{N_s} P_s},
\]

where \(P_n\) are the protein probabilities (describe below). The number of sibling peptides (NSP), which is defined as the expected number of other peptides that correspond to the
Chapter 2. Background on Proteomics Analysis

same protein, is computed as follows:

$$NSP_i^n = \sum_{\{m \neq i\}} w_m^n p(+|D_m),$$ (2.5)

where peptide $m$ is another distinct peptide associated with the protein of interest and $p(+|D_m)$ is the maximum probability of all assignments of peptide $m$ in the dataset. The model simplifies the calculation by putting NSP values into bins. The probability that a correct peptide has an NSP value in bin $k$ is given as

$$p(NSP|+) = \frac{1}{Np(+)} \sum_n \sum_{\{i, NSP_i^n \in k\}} w_i^n p(+|D_i, NSP_i^n),$$ (2.6)

where the prior probability of a correct peptide assignment $p(+)$ is

$$p(+) = \frac{1}{N} \sum_i p(+|D_i, NSP_i).$$ (2.7)

Given $D$ and $NSP$ the probability that peptide assignment is correct is

$$p(+|D, NSP) = \frac{p(+|D)p(NSP|+)}{p(+|D)p(NSP|+) + p(-|D)p(NSP|-)}.$$ (2.8)

The probabilities for the incorrect predictions are calculated in a similar fashion to their correct prediction counterparts. The above equations are iteratively updated using the EM algorithm (Sec. 3.2.5) until a fixed point is reached. Finally, the model computes the protein probabilities according to the following:

$$P_n = 1 - \prod_i (1 - w_i^n p(+|D_i, NSP_i^n)).$$ (2.9)

ProteinProphet scores each candidate protein by taking into account the evidence from its associated peptide predictions. It is considered to be the standard method for making protein predictions. Recently published methods, MSpresso (Sec. 2.6.2) and MSnet (Sec. 2.6.3), proposed algorithms to improve on ProteinProphet by incorporating additional prior knowledge.
2.6.2 MSpresso

Although gene expression data are often employed in areas such as protein function predictions [100, 101, 102, 103, 104], their uses in spectral analyses are a novelty first introduced in the algorithm MSpresso [11]. The algorithm scores each protein prediction by combining the likelihood of the protein’s presence from direct (protein predictions) and inferential (gene expression data from microarray studies) evidence in a Bayesian probability framework. The method leverages the added gene expression information to improve on results from a protein prediction algorithm (ProteinProphet (Sec. 2.6.1) was used in the original study). Both gene expression data and protein predictions are used as its initial inputs.

The core of this algorithm is the MSpresso score, which combines protein prediction scores and gene expression levels. The score is calculated as the probability that a protein is present ($K = 1$) given its observed gene expression (mRNA expression) $M$ and its raw protein prediction score $S$, given as

$$p(K|S, M) = \frac{p(K|S)p(K|M)/p(K)}{\sum_{K=0,1} p(K|S)p(K|M)/p(K)},$$

where $p(K|S)$ and $p(K|M)$ are the posterior probabilities of a protein’s presence given each individual evidence, and $p(K)$ is the prior probability of a protein’s presence. Given $S$, the posterior probability $p(K|S)$ is learned using a logistic regression classifier on $S$. Using a reference protein dataset as ground truth, the log-scale microarray expression value $M$ is first binned; then, for each bin, the posterior probability $p(K|M)$ is set to the fraction of proteins present in the reference protein database. In the original study, $S$ and $p(K|S)$ was set to scores and probabilities output from ProteinProphet, respectively; the protein dataset used is a composition of several published yeast studies [105, 106, 107, 1]; and the microarray expression was the average expression taken from [108, 109, 110]. A fixed prior probability $p(K) = \frac{2}{3}$ was used.

MSpresso has multiple drawbacks. First, it only considers proteins with corresponding
gene expression and discards all other protein predictions. This limits its usefulness, since in practice, mapping proteins to microarray probes are incomplete. In addition, the algorithm relies on a separate algorithm to provide an initial list of protein predictions but does not make additional predictions. Therefore, it is heavily dependent on the quality of the initial protein prediction list. Moreover, comparable microarray data and MS experimental data are hard to find, and proteins can map to multiple microarray probes and vice versa. Lastly, it does not properly account for peptides mapping to more than one proteins.

\section{MSnet}

Similar to MSpresso (Sec. 2.6.2), MSnet \cite{10} takes in a set of protein predictions and re-scores them using protein annotation as prior knowledge; the prior knowledge is the gene functional network (GFN) \cite{98,99}. The GFN, developed for yeast and human, provides a probabilistic framework that integrates annotation from Gene Ontology \cite{111}, and published protein-protein interactions, genetic interactions, phylogenetic profiles and co-citation studies. The use of GFN allows MSnet to reevaluate the likelihood of a protein presence by functional association between proteins. MSnet assumes proteins that are functionally linked are more likely to be co-expressed, and the probabilities of their presence should reflect this information.

The GFN is represented as a graph $G = (V, E)$ with vertices $V$, $|V| = N$ proteins and weighted edges $E$ between nodes where the weight of an edge $w_{ij}$ is the GFN score between proteins $i$ and $j$. Using the network structure in GFN, MSnet computes a score $y_i$ for each protein $i$ that represents the likelihood $i$ is present in the sample given its MS evidence and its functionally linked proteins $J$, which are represented as immediate neighbours in the graph. The score $y_i$ is computed as a convex combination of $i$'s initial protein prediction score $o_i$ and the weighted average of MSnet scores $y_j$ of its immediate neighbouring proteins $j$, $j \in J$. The similarity between $i$ and $j$ is the normalized GFN
score $w_{ij}$ across all scores between $i$ and $J$, given as

$$u_{ij} = \frac{w_{ij}}{\sum_{j:(i,j) \in E} w_{ij}}. \quad (2.11)$$

Since $y_i$ is defined in terms of $y_j$, the algorithm updates the scores iteratively. At $t+1$ iteration the score $y_{i}^{t+1}$ is

$$y_{i}^{t+1} = \gamma o_i + (1 - \gamma) \sum_{j \in J} u_{ij} y_{j}^{t}, \quad (2.12)$$

where the parameter $\frac{(1-\gamma)}{\gamma}$ weights the network’s contribution in the MSnet scores. $\gamma$ is chosen to be a value that optimizes the Area under ROC Curve (AUC) on a reference ‘gold standard’ dataset specific to the organism of interest. Given a setting for $\frac{(1-\gamma)}{\gamma}$, the algorithm is proven to converge.

Like MSpresso, the authors reported a significant improvement over ProteinProphet (8-39%) based on the number of proteins identified at a false positive rate of 5% for the reported yeast dataset. To achieve the results, the authors set $\frac{(1-\gamma)}{\gamma} = 6$, which indicates that the algorithm relies heavily on the prior knowledge. This suggests that the inclusion of prior knowledge can help improve the quality of protein predictions.

There is a number of issues with MSnet. First, the algorithm assumes a complete GFN. Based on Eq. 2.12, it penalizes proteins with missing or no entries in the GFN (assuming $\frac{(1-\gamma)}{\gamma} > 0$ is used), i.e., $u_{ij} = 0$, ($\forall j \in J$). Many proteins in higher organisms, such as mouse and human, are either poorly annotated or have no annotations. This implies that the algorithm has limited applicability in studies based on higher organisms and can only be used to detect known, well-annotated proteins (regardless of the organism). Second, gold standard datasets are generally not available for many organisms, cell types or disease conditions, especially those of higher organisms since their proteome has not completely explored. Therefore, training the algorithm becomes problematic. The tests on the yeast dataset used by the authors do not reflect these problems because the yeast proteome is well studied and the yeast GFN covers most (95%) of the yeast proteome.
teome. Despite its drawbacks, MSnet represents the state-of-the-art protein identification algorithm. Furthermore, it highlights the benefit of incorporating prior knowledge.

The goal of an MS analysis is to discover the identity of the proteins in a complex mixture. The sequence of algorithms consisting of Sequest, PeptideProphet, ProteinProphet and MSnet used in [10] is one such analysis pipeline that is aimed to achieve this goal. Nonetheless, there are two glaring omissions in this pipeline: PTM predictions and PTM refinement. We believe the inclusion of PTM predictions and PTM refinement in the pipeline can benefit from having an improve rate of protein identification and an extra layer of information with the detection of PTMs.
Chapter 3

Background on Machine Learning Concepts

Machine learning (ML) is a discipline in the field artificial intelligence that involves designing and developing models and algorithms which allow computers to automatically learn to recognize complex patterns and make intelligent decisions based on the input (empirical) data. A model is an abstract representation of the problem of interest that describes the intricate relationships between observables (e.g., input data) and hidden causes that gave rise to the observables. Given a model, an ML algorithm is used to learn and infer the optimal setting of model parameters based on the input data. Therefore, a general ML approach to solving a problem begins by designing a model and an appropriate ML algorithm, followed by learning the model using the ML algorithm given the input data. The learned model can then be used to answer questions about the problem. ML has been successful in solving many complex real-world problems that include speech denoising and recognition [112,113], object recognition in video [114], image processing [115] and problems in computational biological, such as gene finding [116], gene and protein functional annotation [103,117,118,3], genetic and physical interaction network analysis [119,120,121], biomarker and motif identifications [122,123], alternative
spicing prediction [124] and PTM predictions [125, 126, 127, 128, 129].

In this chapter, we survey ML concepts used in this thesis. Before getting into the details of the types of ML algorithms, we will begin by presenting graphical representations that are used to effectively represent ML models. Our focus will be on two types of representations, Bayesian networks and factor graphs. Next, we will discuss the different classes of ML algorithms and highlight one such class, clustering. Clustering aims to discover hidden relationships between input data and organize the data into groups in an unsupervised manner. To highlight the differences and benefits of various algorithmic approaches to clustering, we feature four methods: 1) k-means clustering, 2) k-medoids clustering, 3) the EM algorithm for learning mixtures of Gaussians (MOG) and 4) affinity propagation (AP). One issue common to ML methods is selecting the optimal model from many possibilities. We will discuss the idea of model selection and detail two general approaches: the parametric approach and the nonparametric approach using Bayesian statistics. Due to their complexity, nonparametric Bayesian approaches necessitate the use of approximate inference, which we will discuss last. One type of approximate inference we will focus on is sampling. These modelling techniques and machine learning concepts form the basis for our proposed PTM refinement and protein identification algorithms.

3.1 Graphical Representation of Probabilistic Models

A probabilistic model is a mathematical representation of the problem of interest. It uses a set of variables to represent the properties of a problem and a set of functions to represent relationships between these variables. Probabilistic models differ from other mathematical models in that their variables are modelled as probability distributions so as to capture uncertainties and noise in the problem. In many real-world applications,
we are only given a limited number of observations and are interested in determining the underlying causes that give rise to these observations. Probabilistic models can be structured in such a way that they model the processes by which the observed data arose through the use of hidden variables. These types of models are referred to as generative models. Here, the observed data are modelled as visible random variables and the unobserved, underlying causes as hidden or latent random variables, and the probabilistic model describes the joint probability distribution over these variables \[130\]. Complex questions about the problem and inter-relationships between variables can then be answered by calculating specific marginal or conditional probabilities. Although other types of probabilistic models, such as discriminative models, exist, we will focus on generative models as they offer the ability to fully capture the probabilistic nature of all the variables in a model and their inter-relationships; this type of models can be use to model wider diversity of and more complex problems.

As an example, in a generative model used to track objects in a sequence of still images, each observed data point may correspond to an image, while the position and orientation of the objects can be modelled as hidden variables \[131\]. Given the input images, the task is to apply a learning algorithm to infer the position and orientation of the objects, i.e., the hidden variables associated with position and orientation.

Often it is advantageous to represent a probabilistic model diagrammatically as a probabilistic graphical model consisting of nodes (or vertices) and edges (or links). The useful properties that this type of graphical representation provides include: the structure of the probabilistic model can be visualized; the properties of the model, like conditional dependencies and independencies, are captured in the structure of the graph; and the factorization of the joint distribution over all variables in the model into a product of factors each depending only on a subset of the variables can be obtained by inspecting the graph, which can simplify the calculation of the joint distribution significantly \[130\]. Here, each random variable is represented as a node and probabilistic relationships be-
Chapter 3. Background on Machine Learning Concepts

between variables are represented as edges. There are three major types of graphical models: Bayesian networks, which are directed; Markov random fields (MRFs), which are undirected; and factor graphs, which can be either directed or undirected. A directed graph is a graph that has edges with directionality where $A \rightarrow B$ can been seen as $A$ causes $B$ in a causality relationship or $A$ is the parent of $B$ is a parent-child relationship. An undirected graph is a graph with undirected edges and is used to capture the conditional dependencies and symmetric relationships between variables in a model. Below we will focus on Bayesian networks and factor graphs, as they are related to the work presented in this thesis. In addition, we will highlight a short hand notation called plates that can simplify the representation of a complex model with many variables. Graphical models can concisely capture the structure of a probabilistic model and form a framework for computing useful probabilities efficiently.

3.1.1 Bayesian Networks

A Bayesian network (also known as Bayes Net or Bayesian belief network) [132] is a directed acyclic graph (DAG) that is used to capture dependencies and independencies embodied in a given joint probability distribution over a set of random variables. A DAG is a graph with directed edges which has no cycles when following the directions of the edges. Variables of the underlying probabilistic model are represented as nodes, and directed edges between nodes indicate causality pointing from the cause (parent node) to the effect (child node). Edges can also be interpreted as dependencies, where a child node is a dependent of its parent nodes. Descendants are nodes that are downstream from the current node when following the direction of outgoing edges. Conversely, ancestors are nodes that are upstream from the current node, i.e., going against the direction of the incoming edges. Non-descendants are nodes that are not descendants, which also include ancestor nodes.

In a Bayesian network, a node is conditionally independent of any combination of
Figure 3.1: Examples of (a) a Bayesian network and (b) its corresponding factor graph. An equivalent factor graph can be constructed for any Bayesian network. However, the reverse is not true since factor graphs can be more specific about the precise form of the factorization of the joint distribution, and therefore, they can describe models that are not representable by Bayesian network. The circles represent variables and the squares represent local functions associated with the attached variables.
non-descendants given its parents, which can be an empty set if the node does not have a parent. Graphically, a missing edge between two variables that are not descendants of each other implies that these variables are conditionally independent of each other given their parents. Taking the analogy of a family tree further, given the genetic code of John’s parents, knowing the genetic code of his grandparents, uncles, siblings, and grandchildren does not affect our prediction of John’s genetic makeup, i.e., John’s parents’ DNA is sufficient to predict John’s and John’s DNA is conditional independent of this grandparents’ DNA given his parents’ DNA. Furthermore, knowing Mary’s genetic makeup does not affect our prediction of John’s genetic makeup if they are not related. This can be restated as John’s genetic makeup is conditionally independent of the genetic code of his grandparents, uncles, siblings, and grandchildren, given the genetic code of John’s parents; and Mary’s genetic makeup is independent of John’s genetic makeup since they are not linked in anyway (i.e., there is a path along the graph that links Mary and John).

A major benefit of conditional independence properties of a graph is that they allow the joint distribution in a Bayesian network to be factorized in terms of local conditional probabilities of the child given its parents. This can greatly simplify the calculation of the joint probability of a model. Formally, the joint probability for a Bayesian network with variables $X = \{x_1, x_2, ..., x_N\}$ can always be written in the following form:

$$P(X) = \prod_{i=1}^{N} P(x_i|x_{\pi_i})$$

where $N$ is the number of nodes and nodes $x_{\pi_i}$ is the set of parents of node $x_i$ in the graph.

In Fig. 3.1a a simple Bayesian network is shown. The graph shows that $x_2$ and $x_3$ both depend directly on $x_1$ (i.e., $P(x_1, x_2) \neq P(x_1)P(x_2)$ and $P(x_1, x_3) \neq P(x_1)P(x_3)$), but they are conditionally independent of each other given $x_1$ (i.e., $P(x_2|x_1) = P(x_2|x_1)P(x_3|x_1)$). Similarly, $x_3$ and $x_4$ are marginally independent (i.e., $P(x_3, x_4) = P(x_3)P(x_4)$), but they are conditionally dependent given $x_5$, i.e., given $x_5$, $x_3$ and $x_4$
are dependent on each other. By applying Eq. 3.1 to the graph in Fig. 3.1a, the joint
probability can be factorized into the following form:

\[ P(X) = P(x_1)P(x_2|x_1)P(x_3|x_1)P(x_4)P(x_5|x_3, x_4), \]

which enable the joint probability to be calculated by multiplying a number of simpler
probability functions.

When the model has clear causal or parent-child relationships then a Bayesian network
is an efficient and concise way to represent it. The major benefits of Bayesian networks
are the factorization of the joint distribution can be obtained directly from inspecting the
graph structure and it is simple to understand. These benefits make Bayesian networks
used predominately when the problem can be modelled with them. Alternatively, a less
frequently used graphical representation, factor graphs, can be used instead of Bayesian
networks without the lost of information.

### 3.1.2 Factor Graphs

Unlike Bayesian networks, factors graphs [133, 134] have the advantage that they can
capture both conditional dependencies and symmetric relationships between variables in
a model. A factor graph is a bipartite graph, which indicates that two types of nodes
and there are no edges between nodes of the same type. In a factor graph, one set of
\( N \) nodes are called variable nodes (generally drawn as circles) corresponding to variables
\( X = \{x_1, ..., x_N\} \), and the other set of \( M \) nodes are called factors (generally drawn as
squares) corresponding to local functions in the model \( F = \{f_1, ..., f_M\} \), which can be,
for example, probabilities of variables or constraints on the values can take. A function
\( f_m \) in a factor node depends only on its neighbouring variable nodes that are directly
connecting to it, indexed by \( N(m) \subseteq \{1, ..., N\} \). Based on these notations, the joint
distribution over the set of variables \( X \) can be written as a product of factors

\[ P(X) = \prod_{m=1}^{M} f_m(x_{N(m)}). \] (3.2)
Chapter 3. Background on Machine Learning Concepts

For example, Fig. 3.1b on p. 41 shows a factor graph that has $N = 5$ variable nodes representing random variables $X = \{x_1, x_2, x_3, x_4, x_5\}$ and $M = 4$ factor nodes representing local functions, where $f_1$ is connected to $x_1$; $f_2$ to $x_1$ and $x_2$; $f_3$ to $x_1$ and $x_3$; $f_4$ to $x_4$; and $f_5$ to $x_3$, $x_4$ and $x_5$ (so the neighbouring indexes are $N(1) = \{1\}$, $N(2) = \{1, 2\}$, $N(3) = \{1, 3\}$, $N(4) = \{4\}$ and $N(1) = \{1\}$, $N(5) = \{3, 4, 5\}$). This along with Eq. 3.2 implies the joint distribution can be factorized as follows:

$$P(X) = f_1(x_1)f_2(x_1, x_2)f_3(x_1, x_3)f_4(x_4)f_5(x_3, x_4, x_5).$$

Using the above example, the set of marginal probability distributions, $P(x_1), P(x_3), P(x_5)$, can be found by summing over all configurations of other variables as follows:

$$P(x_1) \propto \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} f_1(x_1, x_2, x_3, x_4, x_5)$$
$$= \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} f_1(x_1)f_2(x_1, x_2)f_3(x_1, x_3)f_4(x_4)f_5(x_3, x_4, x_5),$$

$$P(x_3) \propto \sum_{x_1} \sum_{x_2} \sum_{x_4} \sum_{x_5} f_1(x_1)f_2(x_1, x_2)f_3(x_1, x_3)f_4(x_4)f_5(x_3, x_4, x_5),$$

$$P(x_5) \propto \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} f_1(x_1)f_2(x_1, x_2)f_3(x_1, x_3)f_4(x_4)f_5(x_3, x_4, x_5).$$

By applying the distributive law, given as

$$\sum_y [\text{factors independent of } y] \times [\text{factors dependent of } y] = [\text{factors independent of } y] \times \sum_y [\text{factors dependent of } y],$$

the marginal computations can be simplified to

$$P(x_1) \propto \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} f_1(x_1)f_2(x_1, x_2)f_3(x_1, x_3)f_4(x_4)f_5(x_3, x_4, x_5)$$
$$= f_1(x_1) \sum_{x_2} f_2(x_1, x_2) \sum_{x_3} f_3(x_1, x_3) \sum_{x_4} f_4(x_4) \sum_{x_5} f_5(x_3, x_4, x_5),$$

$$P(x_3) \propto \sum_{x_1} \sum_{x_2} \sum_{x_4} \sum_{x_5} f_1(x_1)f_2(x_1, x_2)f_3(x_1, x_3)f_4(x_4)f_5(x_3, x_4, x_5)$$
$$= \left[ \sum_{x_1} f_1(x_1)f_3(x_1, x_3) \sum_{x_2} f_2(x_1, x_2) \right] \times \left[ \sum_{x_4} f_4(x_4) \sum_{x_5} f_5(x_3, x_4, x_5) \right],$$

$$P(x_5) \propto \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} f_1(x_1)f_2(x_1, x_2)f_3(x_1, x_3)f_4(x_4)f_5(x_3, x_4, x_5)$$
$$= \sum_{x_1} f_1(x_1) \sum_{x_2} f_2(x_1, x_2) \sum_{x_3} f_3(x_1, x_3) \sum_{x_4} f_4(x_4) f_5(x_3, x_4, x_5).$$
The above simplification shows that the brute-force approach of summing together all possible products Eq. 3.3 can be reduced to Eq. 3.4, which is more efficient to calculate. This forms the basis for efficient inference algorithms based on message passing for factor graphs and other graphs, which we will discuss later in Sec. 3.2.7.

Bayesian networks and MRFs can be readily converted into factor graphs without loss of information, although the reverse is not necessarily true. One method to convert a Bayesian network to a factor graph is as follows. First, we create a variable node for each variable node in the Bayesian network, and then create a factor node for each conditional distribution in the model. Next, for each variable node associated with a conditional distribution, we create an edge between it and the corresponding factor node. Repeat this for all conditional distributions in the model. It is important to note that there can be multiple factor graphs that correspond to the same Bayesian network. An example of the conversion of a Bayesian network to a factor graph is illustrated in Fig. 3.1 on p.41.

Similarly, to convert an MRF to a factor graph, a variable node is created for each variable node in the original MRF, and then a factor node is created for each maximal clique. A clique is defined as a set of fully-connected nodes in the graph, and a maximal clique is defined as a clique such that it cannot be expanded by additional of other nodes and remain a clique. The local function for each factor node is set to its corresponding clique potential, the joint distribution that describes the clique. Again, there can be several different factor graphs that correspond to the same MRF.

When compared to both Bayesian networks and MRFs, factor graphs include additional factor nodes that make them structurally more complex but allows calculations, such as marginal distributions, to be factorized more precisely and efficiently computable. Given their generality, factor graphs are often used to describe a class of efficient inference methods, i.e., message passing algorithms. Two such algorithms, the sum-product and max-product algorithms, are discussed later in Sec. 3.2.7. The key feature in these
message passing algorithms is that the number of operations required for calculating all conditional and marginal probabilities are the same as those for calculating conditional and marginal probabilities for only two of the variables. They make use of the factorization property of a model drawn up in the form of a factor graph.

The benefits of factor graphs over the other types of graphical representations enable factor graphs to represent not only models that Bayesian networks or MRFs can describe, but also a number of additional models not representable by Bayesian networks and MRFs. Factor graphs are particular appealing for problems that require complex models that are difficult or impossible to describe with either Bayesian networks or MRFs. When a model contains many variables, it can be further simplify with a shorthand notation using plates.

3.1.3 Plate Notation

Regardless of the type of graphical representation, when dealing with large, complex graphical models involving interactions between many variables, it is often convenient to represent the models in plate notations. A plate notation involves the simplification of a graphical model where nodes that are replicated graphically multiple times are represented by an instance of the duplicated nodes drawn inside a box (plate) that indicates the number of times the nodes inside are duplicated. Fig. 3.2(a) and Fig. 3.2(b) show an example of a Bayesian network in its explicit form and in its plate notation, respectively. The repeated groups (nodes $b_1, b_2, b_3, ..., b_N$ and $c_1, c_2, c_3, ..., c_N$) are replaced by nodes $b_n$ and $c_n$ with a box around them. The box is referred to as a plate, and it implies that the enclosed subgraph is repeated $N$ times and edges linking between nodes outside the plate to nodes inside the plate are duplicated. Although the plate notation can help simplify (graphically) a complex-looking graphical model, its drawback is that it cannot represent the inter-dependencies between similar nodes. Expending on our example, if $b_i$ is a child of $b_{i-1}$ (i.e., there is an edge between $b_i$ and $b_{i-1}$) then the plate
Figure 3.2: Example of (a) a Bayesian network drawn up explicitly and (b) in plate notation. The box in (b) is called a plate. The enclosed subgraph is repeated $N$ times and edges linking between nodes outside the plate to nodes inside the plate are duplicated. The plate notation allows a simplified representation of the graphical model.
notation cannot adequately capture these inter-dependencies.

### 3.2 Machine Learning (ML) Algorithms

Above we have presented various ways to represent probabilistic models. Next, we will discuss the uses of these models. The use of probabilistic models involves identifying the optimal parameter settings for the mathematical functions used in the model, also known as hypothesis space selection. This can be done by either setting the optimal parameter settings based on expert knowledge or learning them inductively from a set of observed data. The former approach requires expertise in the problem, which is not often available in many real-life applications. Alternatively, the latter approach takes the observed data and learns the optimal parameter settings that best describe the features in the dataset and their interrelationships. Algorithms that perform this type of learning are called machine learning (ML) algorithms [130]. This type of algorithms is best suited to solve problems where the underlying patterns are not well understood but can be characterized by a dataset with strong statistical regularity.

Generally, learning involves coming up with a set of models and identifying the optimal parameter settings for them on a set of input training data. The principal objective of learning is to identify a model and parameter settings that can achieve the best predictive performance on new data or succinctly describe the training data. It is common to evaluate models on an independent labelled test dataset that was not used to learn the model. However, when the labels on the test dataset are unknown, evaluation of the learned models is difficult and models may be assessed based on expert knowledge or known ground truth.

There are three main types of ML algorithms: 1) supervised learning, 2) unsupervised learning and 3) reinforcement learning. The use of each type of algorithm depends both on the problem of interest and the information available in the dataset, such as labels on
the training data.

Supervised learning is a class of algorithms that is concerned with making predictions of a target value based on a set of input features (i.e., input data) [130, 133]. Supervised learning can be used to perform classification when the target is discrete, such as classifying incoming email into spam and not spam [135]; and can be applied to regression problems, such as stock price prediction [136], where the outputs are continuous values. A limitation of supervised learning is that it requires a set of training data with labels, such as known spam emails or stock prices, thus restricting it to only problems with well-defined input and output pairs.

Unsupervised learning is a class of algorithms that is concerned with extracting ‘meaningful’ hidden causes for a set of input data [130, 133]. It differs from supervised learning in that the training data do not have corresponding target outputs to guide the learning. Unsupervised learning can be used to partition a set of data into groups of similar examples, commonly known as clustering. A goal of clustering is to automatically uncover structures that lay hidden in the input data. As an extension to the idea of clustering, unsupervised learning algorithms can be applied to perform outlier detection, where the input data are analyzed to find and separate highly unusual cases. Another application in unsupervised learning is factorization / dimensionality reduction. Here, the goal is to provide an encoder and decoder pair such that the encoder transforms the high dimensional input data into a low dimension representation, e.g., down to one or two dimensions; and the decoder converts the output from the encoder back to the original input data with minimal noise and information loss. This approach has been applied in advanced image and video processing research, where, for example, the goal is to learn a patch-based model that provides an epitome (a summary or a prototype), which captures interesting properties for the input images (or videos) [137, 138].

The last type of ML algorithms, reinforcement learning, is a class of algorithms that is concerned with the problem of finding the best set of actions in a given situation
which will maximize the reward \[139\]. The set of actions (inputs) that yield the optimal reward (labelled output) are not given, in contrast to supervised learning, but reinforcement learning uses a process of trial and error to discover them. For example, the game of backgammon can be mastered by learning a neural network using appropriate reinforcement learning techniques \[140\]. Here, the neural network must take in the board position and the result of the dice throw as input and produce a ‘strong’ move as an output. The key challenge here is that the game involves multiple moves but the reward is only available at the end of the game, in the form of victory. Therefore, the algorithm must attribute the reward appropriately to the series of moves that led to it, even though some moves might be considered as bad moves; and it must apply the ‘knowledge’ learned in all the games previously played to the current game. The key feature in reinforcement learning is the trade-off between exploration, where the algorithm tries new moves to see how effective they are, and exploitation, where the algorithm takes the moves known to yield a high reward. Therefore, the goal is to learn a model that finds the right balance between exploration and exploitation as a strong bias towards either exploration or exploitation is known to yield poor results.

Many problems in proteomics lack complete, reliable, unbiased data that can be used to supervise learning. We have already discussed two examples of such problems in Ch. 2: PTM refinement and protein prediction from MS/MS experiments. The field of PTM studies has only just begun, and data on known PTMs and PTM sites are scarce. Therefore, an unbiased dataset that has a representative coverage of the PTM space is presently not available to guide the learning. Similarly, mapping from spectra to peptide sequences to proteins is imprecise. This is supported by the facts that only a small fraction of spectra can be confidently mapped to peptides (\(\sim 20\%\)) \[9, 10, 11\] and there is a lack of overlap between results from different spectral analysis algorithms \[9, 10, 11, 84, 8\], which suggest a comprehensive unbiased dataset for training a model is not available. Given these settings, a prefer ML approach is an unsupervised method for many problems
Chapter 3. **Background on Machine Learning Concepts**

in proteomics.

Below we will describe in more detail clustering, which is a type of unsupervised learning that is applicable to many problems in proteomics and other areas of biology. To illustrate how clustering works, we present two simple non-probabilistic methods, \( k \)-means and \( k \)-medoids clustering, and two powerful probabilistic methods: the EM algorithm applied to MOG, which builds on the concept of \( k \)-means clustering; and AP using loopy belief propagation message passing algorithm, which, like \( k \)-medoids clustering, is an exemplar-based clustering (EBC) algorithm. Throughout this section, we will prelude each clustering method with the appropriate parameter optimization and inference algorithm when necessary. For the probabilistic methods, an important part of the learning algorithms is the method used to optimize the parameters. In the absence of ground truth, two goals (objectives) that many parameter optimization methods use are maximum likelihood (ML) and maximum *a posteriori* (MAP) estimations. Lastly, we touch on the concept of model selection, where the goal is to find the best learned model amongst a set of all possible models. We detail two general approaches to model selection: the parametric approach and the nonparametric approach using Bayesian statistics.

### 3.2.1 Clustering

Clustering is one of the fundamental methods for data analysis, where the goal is to partition the data into meaningful clusters. Object detection in images \[141\] is an example of clustering where input images are grouped into clusters of similar images. Another example is in advanced communication research, where, for example, the goal is to design efficient systems to transmit complex signals, such as speech and video, through a limited bandwidth channel, such as telephone or Ethernet cables, by quantizing (i.e., clustering) the signal into discrete subspace of lower dimension \[142\] \[143\].

Exemplar-based clustering (EBC) is a special type of clustering approach that, in addition to partitioning the data, identifies a prototypic data point within each cluster
Chapter 3. Background on Machine Learning Concepts

52

(exemplar). EBC problems arise naturally in diverse fields and EBC has been applied successfully in a number of areas, such as visual scene analysis [144], image segmentation [145] and the analysis of chemical-genetic interaction data and HIV vaccine design [146]. Where EBC is applicable are when exemplars have meaningful interpretations and finding them, instead of cluster assignments, is the objective of the analysis. Below, we will give specific examples of various clustering algorithms to highlight their differences, strengths and weaknesses.

3.2.2 Example: \textit{k}-means Clustering

One of the simplest, non-probabilistic unsupervised learning algorithms is the \textit{k}-means clustering algorithm [147], where each data point is assigned to one of \( K \) (predefined constant) clusters based on a predefined similarity measure. Here, each cluster is represented by its centre or mean. Like all learning algorithms, the aim in \textit{k}-means clustering is to minimize an objective function, which, in this case, is the sum of squared distance between each data point and its cluster centre. This can be achieved by finding the optimal setting for the cluster centres and the assignment of data points to the clusters. The objective function for \( N \) data points is given as

\[
J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2 ,
\]

where \( r_{nk} \in \{0, 1\} \) is a binary indicator such that \( r_{nk} = 1 \) when data point \( x_n \) is assigned to cluster \( k \) and \( r_{nk} = 0 \) otherwise; and \( \|x_n - \mu_k\|^2 \) is the squared Euclidean distance between data point \( x_n \) and mean \( \mu_k \).

As an example, Fig. 3.3 illustrates the application of \textit{k}-means clustering to the Old Faithful dataset [148], represented as a set of data points in a two-dimensional space. To start, the set of means \( \mu_k \), one for each cluster, are set randomly to some initial values. While keeping \( \mu_k \) fixed, each data point is assigned to the cluster with the nearest mean. More formally, this can be expressed as minimizing \( J \) with respect to \( r_{nk} \) by setting \( r_{nk} \)
Figure 3.3: Illustration showing the result of applying the $k$-means clustering algorithm to cluster a set of data points (re-scaled Old Faithful dataset [48]) into two groups at various iterations of the algorithm. The cluster centres are represented by "*" and the colour of the data points (red or blue) represents their cluster assignment. In this example, $K = 2$. 

Figure 3.3: Illustration showing the result of applying the $k$-means clustering algorithm to cluster a set of data points (re-scaled Old Faithful dataset [48]) into two groups at various iterations of the algorithm. The cluster centres are represented by "*" and the colour of the data points (red or blue) represents their cluster assignment. In this example, $K = 2$.
to 1 for whichever value of \( k \) that gives the minimum value of \( \|x_n - \mu_k\|^2 \), and 0 to all other \( k \)'s. After all the data are assigned, each \( \mu_k \) is recalculated to be the mean of the data in its cluster. This can be seen as minimizing \( J \) with respect to \( \mu_k \) by setting the derivative of \( J \) with respect of \( \mu_k \) to zero, giving

\[
2 \sum_{n=1}^{N} r_{nk}(x_n - \mu_k) = 0,
\]

and solving for \( \mu_k \) to give

\[
\mu_k = \frac{\sum_n r_{nk}x_n}{\sum_n r_{nk}}. \tag{3.6}
\]

The algorithm continues to iterate between updating \( r_{nk} \) and recalculating \( \mu_k \) until the \( \mu_k \)'s no longer change or until some maximum number of iterations elapse. Intuitively, this is equivalent to minimizing \( J \) with respect to \( r_{nk} \) and \( \mu_k \) until convergence. In the given example, as the means get updated, the data points are reassigned to either one of the two clusters, where \( K \) is set to 2.

The performance of this algorithm depends significantly on the choice for \( K \) (how this can be done will be discussed later in Sec. 3.2.9) and on the initial setting of the means. This means this algorithm is highly prone to local minima. Depending on the problem and the distribution of the input data, various heuristics for initializing the means have been proposed. These heuristics include evenly distributing the means across the range of data points, or rerunning the algorithm with different random initializations of the means each time and choosing the solution with lowest \( J \). The \( k \)-means clustering algorithm has been applied to a number of problems in computation biology with success, such as protein family and superfamily predictions [149] and protein quantitative analysis [150]. However, due to its non-probabilistic nature, it cannot be applied to solve complex problems that require the use of hidden variables.
Figure 3.4: Illustration showing the result of applying the \( k \)-medoids clustering algorithm to cluster the re-scaled Old Faithful dataset [148] into two groups at various iterations of the algorithm. The exemplars found at each iteration are represented by “*” and the colour of the data points (red or blue) represents their cluster assignment. In this example, \( K = 2 \).

### 3.2.3 Example: \( k \)-medoids Clustering

Similar to \( k \)-means clustering, \( k \)-medoids finds \( K \) partitions of the data points. It differs from \( k \)-means clustering by finding exemplars instead of means and allowing arbitrary data similarity measures instead of squared distance between each data point and its cluster centre. As an example, negative distances can be used as similarity measures.

Given \( N \) data points and data similarities \( s(n, k) \) between pairs of data points \( n \) and \( k \) \((n, k \in \{1, ..., N\})\), the objective for \( k \)-medoids clustering is to identify \( K \) exemplars and assign each data point to an exemplar so to minimize the sum of negative data similarities (equivalent to maximizing the sum of data similarities) between each data
Chapter 3. Background on Machine Learning Concepts

point and its exemplar. The objective function is given as

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} -r_{nk}s(n,k),$$

(3.7)

where $r_{nk} \in \{0,1\}$ is a binary indicator, such that $r_{nk} = 1$ when data point $n$ is assigned to exemplar $k$ and $r_{nk} = 0$ otherwise.

To illustrate, Fig. 3.4 shows the application of $k$-medoids clustering on the Old Faithful dataset used previously when discussing $k$-means clustering in Sec. 3.2.2. The algorithm initializes by randomly picking a set of data points as exemplars $\kappa$, where $|\kappa| = K$ and $K = 2$ in this example only. At each iteration, data points are assigned to an exemplar $k$ ($k \in \kappa$) most similar to them. More formally, this can be expressed as minimizing $J$ with respect to $s(n,k)$ by setting $r_{nk} = 1$ for any value of $k$ that minimizes $-s(n,k)$, and $r_{nk} = 0$ for all other $k$’s. Then, for each cluster, set the exemplar to a data point $j$ that is most similar to all other data points within that cluster, which can be written as

$$\forall k \in \kappa : k \leftarrow \arg \max_{j:r_{jk}=1} \sum_{n=1:r_{nk}=1 \text{ and } n \neq j} s(n,j).$$

(3.8)

The algorithm continues to iterate until the set of exemplars $\{r_{nk}\}$ no longer changes or until some maximum number of iterations elapse.

In this example, the resulting cluster assignments from $k$-means and $k$-medoids clustering are almost identical. However, in general, the results from $k$-means and $k$-medoids clustering can be very different but neither outperforms the other. The decision to use one over the other is based on the application and the desire output. The $k$-medoids clustering algorithm is preferred if exemplars have specific meanings and finding them is the goal of the analysis. Similar to $k$-means clustering, $k$-medoids clustering requires the number of clusters $K$ as an input and cannot be applied to solve complex problems that need the use of hidden variables.
3.2.4 ML and MAP Estimations

In contrast to non-probabilistic clustering methods like the ones discussed above, probabilistic clustering algorithms can account for uncertainties in the data and clustering assignments. As a result, probabilistic clustering algorithms better model the problem. Before we introduce probabilistic clustering algorithms, we will present the general approaches used for model inference. One of the major tasks in learning probabilistic models is inferring the most probable configuration of model parameters that best describe the system of interest. Two common ways to achieve this are by the maximization of either the data likelihood (ML estimation) or the posterior probability (MAP estimation). The posterior distribution can be seen as our belief of the model parameters given our prior knowledge of these model parameters and after the likelihood of the data is taken into account. Formally, it is proportional to the prior probability (or simply prior) times the data likelihood. In this section we first introduce the method of inferring the most probable configuration of model parameters by ML estimation. Second, we will discuss inference by MAP estimation.

In a model represented using a Bayesian network, the likelihood of the parameters $L(\theta)$ given the set of $N$ data points is defined as

$$L(\theta) = \prod_{n=1}^{N} \sum_{h_n} P(h_n, v_n | \theta),$$

where $v$ are the visible variables, $h$ are the hidden variables, and $\theta$ are the model parameters. The likelihood function expresses how probable the observed dataset is for the parameter setting $\theta$. The summation over all the hidden variables is required to account for the unknown and dependent nature of the hidden variables. For continuous hidden variables, the summation over the hidden variables is replaced by an integration. For simplicity, we will consider only models with discrete hidden variables in our discussion though the concepts extend to models with continuous variables as well.

One way of determining the optimal setting of $\theta$ is by application of ML estimation,
in which the value of \( \theta \) that maximizes the probability of the observed dataset is chosen, i.e., set

\[
\theta = \arg \max_{\theta} L(\theta), \quad (3.10)
\]

\[
= \arg \max_{\theta} \prod_{n=1}^{N} \sum_{h_n} P(h_n, v_n|\theta).
\]

To avoid numerical underflow, all calculations are commonly performed in the logarithmic domain, such that the log likelihood is optimized instead of the likelihood. The log likelihood has the following form:

\[
\ln L(\theta) = \sum_{n=1}^{N} \ln \sum_{h_n} P(h_n, v_n|\theta). \quad (3.11)
\]

To obtain the optimal setting for each parameter, the derivative of the log likelihood with respect to each parameter is set to zero and resulting set of equations may then be used to solve for parameters. In machine learning, the negative log likelihood function is referred as an error function \([130]\). Since the negative logarithm is strictly monotonic and decreasing, maximizing the likelihood has the interpretation of minimizing the error.

In cases where prior knowledge of the model parameters is available, we can incorporate this information as a prior probability of the parameters, \( P(\theta) \). Here, instead of determining the ML parameters, we can estimate the MAP parameter setting, which, with proper prior knowledge, is a more meaningful evaluation of the fitness of the model. By using Bayes rule we can calculate the posterior distribution of the parameters and obtain the MAP setting as follows:

\[
\theta = \arg \max_{\theta} \prod_{n=1}^{N} P(\theta|v_n), \quad (3.12)
\]

\[
= \arg \max_{\theta} \prod_{n=1}^{N} \sum_{h_n} P(h_n, v_n|\theta) P(\theta) \int_{\theta} \prod_{n=1}^{N} \sum_{h_n} P(h_n, v_n|\theta) P(\theta)
\]

\[
= \arg \max_{\theta} \prod_{n=1}^{N} \sum_{h_n} P(h_n, v_n|\theta) P(\theta).
\]
The denominator $\int \prod_{n=1}^{N} \sum_{h_n} P(h_n, v_n|\theta) P(\theta)$ is not needed here since it remains constant for all $\theta$ and does not affect the answer. In cases where the prior distribution on the parameters is uniform, ML and MAP yield the same result.

When the model does not contain any hidden variables the ML and MAP estimation can be derived analytically or approximated using gradient-base optimization techniques, such as gradient descent or conjugate gradients [151]. However, these methods often cannot be directly applied to models with hidden variables because the summation over the hidden variables is intractable, making exact inference unattainable. To account for the presence of hidden variables, an approximate method like the EM algorithm (Sec. 3.2.5) is often used to perform ML estimation. In the case of MAP estimation, approximate methods like variational inference [152, 153, 154, 155], Expectation Propagation (EP) [156] and Monte Carlo sampling [157, 158] are generally used. In this thesis, we focus on a particular type of Monte Carlo sampling methods, Markov chain Monte Carlo (MCMC), which we discuss later in Sec. 3.2.10.

3.2.5 The Expectation-Maximization (EM) Algorithm

Before discussing the probabilistic clustering algorithm, mixture of Gaussians (MOG), we will introduce the inference algorithm that is often used for MOG, the expectation-maximization (EM) algorithm [159, 160]. The EM algorithm is an elegant and powerful method for finding the maximum likelihood of models with hidden variables. The key concept in the EM algorithm is that it iterates between the E-step and M-step until convergence. In the E-step, the algorithm estimates the posterior distribution $Q$ of the hidden variables given the observed data and the current parameter settings ($Q$ is also referred to as the responsibility); and in the M-step, the algorithm calculates the ML parameter settings with $Q$ fixed. At the end of each iteration, the lower bound on the likelihood is optimized for the given parameter setting (M-step) and the likelihood is set to that bound (E-step). The process guarantees an increase in the likelihood and
convergence to a local maximum, or to a global maximum if the likelihood function is unimodal.

Formally, for \( N \) input data points at iteration \( t \), given the current parameter settings \( \theta^{t-1} \) calculated in the previous iteration, the E-step estimates the responsibility \( Q \) for the \( n \)-th input data point as

\[
Q_{nh_n} = P(h_n|v_n, \theta^{t-1}) = \frac{P(h_n, v_n|\theta^{t-1})}{\sum_{h_n} P(h_n, v_n|\theta^{t-1})}. \tag{3.13}
\]

\( Q \) is estimated by the probability of the hidden variables given the input data and the current setting of the model parameters. In the M-step, while fixing \( Q_{nh_n} \), the parameters are estimated by maximizing the expected complete log likelihood

\[
\theta^t = \arg \max_{\theta} E_{h_n} \left[ \sum_n \ln P(h_n, v_n|\theta) \right], \tag{3.14}
\]

\[
= \arg \max_{\theta} \sum_n \sum_{h_n} Q_{nh_n} \ln P(h_n, v_n|\theta),
\]

where \( E_{h_n} \left[ \sum_n \ln P(h_n, v_n|\theta) \right] \) is the expectation of the log likelihood with respect to the hidden variables \( h_n \) for the \( n \)-th data point. The EM algorithm can be altered to calculate MAP instead of ML parameter settings by switching the ML estimation in the M-step with MAP estimation, where the posterior distribution is maximized instead the log likelihood.

As an example, the EM algorithm is applied to learn an MOG model for clustering the set of data points used in Sec. 3.2.6. The derivation of the E-step and M-step for the MOG is given in Appendix A.

### 3.2.6 Example: Mixture of Gaussians (MOG)

There are many cases where a hard assignment of data points to clusters, such as in the case of \( k \)-means clustering, is not reasonable due to cluster uncertainty and noise associated with the data. Mixture models attempt to cluster the data using a soft
assignment based on the posterior probability distribution of the cluster assignment, given the observed data points. The idea is to allow different sub-models to contribute to the density model in different parts of the space while allowing multiple models to overlap in some regions. In the case of clustering using a mixture of Gaussians (MOG) model, where the $K$ clusters are modelled as Gaussian distributions with parameters mean $\mu_k$, variance (or covariance for multidimensional dataset) $\Sigma_k$ and mixing coefficient (or weight) $\alpha_k$, the likelihood of the data is the weighted contribution of the $K$ Gaussian distributions. In the following discussion, we will consider the case where $\Sigma_k$ is the variance with a scalar value instead of the covariance. The Bayesian network of an MOG model for a set of $N$ data points $x_n$ and corresponding hidden variables $z_n$ representing the cluster assignment for each data point is shown in Fig. 3.5.

Formally, the probability of $z_n$ taking on a value $k \in \{1, ..., K\}$ is a multinomial distribution given as

$$P(z_n = k) = \alpha_k. \quad (3.15)$$
The conditional distribution of $x_n$ given a particular value of $z_n$ is given as

$$P(x_n|z_n = k) = \mathcal{N}(x_n|\mu_k, \Sigma_k),$$

(3.16)

where the Gaussian distribution has the form

$$\mathcal{N}(x_n|\mu_k, \Sigma_k) = \frac{1}{\sqrt{2\pi\Sigma_k}} \exp\left\{-\frac{1}{2\Sigma_k}(x_n - \mu_k)^2\right\}.$$

(3.17)

Using Eq. 3.1, the joint distribution can be factorized as $P(z)P(x|z)$. Then, from Eq. 3.14, the expected log likelihood function for the MOG model can be written as

$$\ln L(\theta) = \sum_n \sum_k Q_{nz_n} \ln P(x_n, z_n|\mu_k, \Sigma_k),$$

(3.18)

$$= \sum_n \sum_k P(z_n = k|x_n, \mu_k, \Sigma_k) \left[ \ln \alpha_k - \frac{1}{2} \ln(2\pi) - \frac{1}{2} \ln(\Sigma_k) - \frac{(x_n - \mu_k)^2}{2\Sigma_k} \right],$$

where mixing coefficient $\alpha_k = P(z_n = k)$ satisfies $\alpha_k \geq 0$ and $\sum_k \alpha_k = 1$, and $Q_{nz_n} = P(z_n = k|x_n, \mu_k, \Sigma_k)$.

Since the MOG model consists of hidden variables, the EM algorithm is used to find the ML solution. In our example of the Old Faithful dataset using a mixture of two Gaussians (Fig. 3.6), $\mu_k$’s are initialized to the means used to initialize the $k$-means clustering example in Sec. 3.2.2, $\Sigma_k$’s for both dimensions are initialized to 1, and the mixing coefficients $\alpha_k$’s are initialized to 0.5. For the sake of simplicity and ease of comparison, in the current example we modelled the data using a scalar variance for each dimension instead of covariance. There is a high similarity between the parameter updates between $k$-means clustering and MOG, which we will highlight below as we outline the E-step and the M-step. The detailed derivation of the E-step and the M-step are given in Appendix A.

In the E-step, the responsibility $Q_{nk}$ is calculated using the data points and the
Figure 3.6: The result of applying the MOG model to cluster a set of data points (re-scaled Old Faithful dataset [148]) into two groups at various EM iterations is shown. The cluster centres are displayed as "*", the oval around the cluster centres indicates the Gaussian standard deviation, and the colour each data point represents its likelihood of belonging to each cluster.
current setting of the parameters, given by

\[ Q_{nk} = P(z_n = k | x_n, \mu_k, \Sigma_k), \]
\[ = \frac{P(z_n = k)P(x_n | z_n = k, \mu_k, \Sigma_k)}{\sum_{j=1}^{K} P(z_n = j)P(x_n | z_n = j, \mu_j, \Sigma_j)}, \]
\[ = \frac{\alpha_{nk}N(x_n | \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \alpha_{nj}N(x_n | \mu_j, \Sigma_j)}. \hspace{1cm} (3.19) \]

The \( Q_{nk} \) is very similar to the \( r_{nk} \) in \( k \)-means clustering, except that \( Q_{nk} \) provides a soft assignment whereas \( r_{nk} \) provides a hard assignment of \( n \)-th data point to the \( k \)-th cluster.

In the M-step, while \( Q_{nk} \) remain fixed, the ML setting of the parameters are calculated by setting the derivatives of the expected complete log-likelihood given in Eq. 3.18 with respect to mean \( \mu_k \) to zero and solving for \( \mu_k \). Thus,

\[ \mu_k = \frac{\sum_{n=1}^{N} Q_{nk}x_n}{\sum_{n=1}^{N} Q_{nk}}. \hspace{1cm} (3.20) \]

This states that the mean of the \( k \)-th component \( \mu_k \) is just the mean of the data points, weighted by the responsibility that the corresponding data points were generated from the \( k \)-th component. The updates for \( \mu_k \) for both MOG Eq. 3.20 and \( k \)-means clustering Eq. 3.6 have a very similar form. The difference is that in MOG \( Q_{nk} \) is used, which provides a soft cluster assignment, whereas in \( k \)-means clustering \( r_{nk} \) is used, which provides a hard cluster assignment. A soft cluster assignment is one that gives a probability for assigning to each cluster, whereas a hard cluster assignment is one that states which cluster the data point is assigned to.

Similarly, setting the derivative of the expected log-likelihood with respect to \( \Sigma_k \) to zero and solving for \( \Sigma_k \), we get

\[ \Sigma_k = \frac{\sum_{n=1}^{N} Q_{nk}(x_n - \mu_k)^2}{\sum_{n=1}^{N} Q_{nk}}. \hspace{1cm} (3.21) \]
This means that the variance of the $k$-th component $\Sigma_k$ is given by the variance of the data points with respect to the mean $\mu_k$ of the $k$-th component, also weighted by the responsibility $Q_{nk}$ that the cluster explains the data points.

Finally, we obtain the update equation for the mixing coefficient $\alpha_k$ by maximizing the expected log-likelihood with respect to $\alpha_k$ and solving for $\alpha_k$ with the following two constraints: the mixing coefficients must sum up to one and they must be strictly positive. Thus,

$$\alpha_k = \frac{\sum_{n=1}^{N} Q_{nk}}{\sum_{j=1}^{K} \sum_{n=1}^{N} Q_{nj}}.$$  \hspace{1cm} (3.22)

Intuitively, $\alpha_k$ for the $k$-th component is simply the responsibility for that component, averaged over all the data points.

In the example (Fig. 3.6), after each iteration the means of the two clusters, indicated by “*”, are repositioned with corresponding variance represented by the ovals, and the colour represents the soft cluster assignment for the data points. One of the strengths of this algorithm is that it provides a likelihood value for the data points belonging to each cluster, which is especially important for analyzing data points that lie between clusters, shown as purple dots. Furthermore, in this model $\Sigma_k$’s are allowed to change to better fit the data, which helps to avoid local optima. For example, if the data points represent peptides with modifications and the task is to cluster them into the different types of PTMs based on their modification masses and location of the modifications, then knowing the likelihood with which each peptide belongs to different types of PTMs is very important.

Due to the similarity between $k$-means clustering and MOG, the learning algorithm for MOG can be easily modified to perform $k$-means clustering by fixing each variance $\Sigma_k$ to be a very small non-zero value and each mixing coefficient $\alpha_k$ to be $\frac{1}{K}$, while not updating them at each EM iteration. Setting $\Sigma_k$ to a small number changes the MOG from making soft assignments to hard assignments. Similarly, maximizing the
expected log-likelihood Eq. 3.18 is equivalent to minimizing $J$ for $k$-means clustering Eq. 3.5. In this setup, the EM algorithm can be used to learn $k$-means clustering by applying the above changes to the algorithm.

While clustering with the MOG model has its strengths, the time it takes to converge is considerably longer than that taken by $k$-means clustering. The number of iterations performed by the EM algorithm for the MOG model is much greater than the number of iterations used in the $k$-means clustering, and within each iteration, the EM algorithm takes significantly more computations. Given that it is common to perform multiple restarts to achieve a good solution, the speed difference between the two algorithms is even more significant. It is therefore a common practice to determine a suitable initial parameter setting by running $k$-means clustering before learning the MOG model with EM.

3.2.7 Inference with Message Passing Algorithms

In Sec. 3.2.5, we presented the EM algorithm as a method to calculate the ML or MAP estimators for graphs with directed edges such as Bayesian networks. Here, we will present a class of inference algorithms based on the idea of message passing that works for both directed and undirected graphs. We will introduce three variants: the sum-product, the max-product and the loopy belief propagation algorithm. The sum-product algorithm \cite{130} performs exact inference by calculating local marginal over nodes or subsets of nodes. When the most probable parameter and variable setting is preferred, e.g., MAP estimation, the sum-product algorithm can be modified to produce the max-product algorithm \cite{130, 161}. Alternatively, a special case of the sum-product algorithm, called belief propagation \cite{132, 162}, has been introduced to work specifically for directed graphs. A significant limitation of both the sum-product and max-product algorithms is that they only work for graphs that are trees, i.e., graphs with no cycles (or loops). For complex graphs with cycles, exact inference is impossible and approximate inference
is used instead. In such cases, loopy belief propagation \[163\] can be applied. To illustrate how message passing algorithms work, we highlight the affinity propagation (AP) algorithm and applied it on the Old Faithful dataset at the end of this section. AP uses the loopy belief propagation algorithm for approximate inference. All three algorithms, sum-product, max-product and loopy belief propagation, are best described using factor graphs.

**Sum-product Algorithm**

Given the joint distribution $P(X)$ of a model expressed as a factor graph, the sum-product algorithm can be used to efficiently find marginal over component variables $X = \{x_1, \ldots, x_N\}$. The algorithm works by passing real values (messages) between variable nodes and factor nodes. These messages can be seen as the ‘influence’ that one variable $x_i$ exerts on another $x_j$ on the value $x_j$ should take on. Therefore, the marginal distribution of a variable can be seen as the probability of taking on a value based on the combined ‘belief’ from all other variables. The marginal distribution can be calculated by summing over messages from all other nodes in the graph. The key to this algorithm is that the marginal distributions of all variables can be calculated in just two passes through the graph, which is equivalent to calculating marginal distributions for two variables. It leverages on the distributive law to factorize the joint distribution into components that rely only on operations on local variables, similar to how marginal computations of variables in a factor graph can be simplified (see Sec. 3.1.2).

There are two types of messages, illustrated in Fig. 3.7: a message from a factor to a variable node and a message from a variable to a factor node. The message sent from a factor node $f_m$ to a variable node $x_n$ is the product of the factor node with all incoming messages from its neighbouring variables $x_{n'}$ marginalized over $x_n$, where $n' \in N(m) \setminus n$ and $N(m) \setminus n$ are all neighbouring variable nodes excluding $x_n$. It reflects the marginal probability distribution about $x_n$ given $f_m$ and the current probability distribution of its
Chapter 3. Background on Machine Learning Concepts

Figure 3.7: The sum-product (or max-product) algorithm messages illustrated on a factor graph. A factor-to-variable message \( \mu \), shown in (a), is computed from incoming variable-to-factor messages \( \nu \)'s along connecting edges. Likewise, a variable-to-function message \( \nu \) in (b) is computed from incoming factor-to-variable messages \( \mu \)'s. The marginal (or MAP estimate) for a variable can be computed from all incoming factor-to-variable messages \( \mu \)'s, shown in (c). Here, \( N(n) \) indicates factor nodes that are neighbours of the variable node \( x_n \), \( N(n) \setminus m \) are factor nodes that are neighbours of \( x_n \) excluding the factor node \( f_m \), and \( N(m) \setminus n \) are variable nodes that are neighbours of the factor node \( f_m \) excluding the variable node \( x_n \).
neighbouring nodes. The message takes the form

\[
\mu_{m \rightarrow n}(x_n) = \sum_{x_{n' : n' \in N(m) \setminus n}} f_m(x_{N(m)}) \times \prod_{n' \in N(m) \setminus n} \nu_{n' \rightarrow m}(x_{n'}), \quad (3.23)
\]

where \(x_{N(m)} \equiv \{x_n\}_{n \in N(m)}\) is the shorthand notation for the set of neighbouring variable nodes to factor node \(f_m\).

The message sent by a variable node \(x_n\) to a factor node \(f_m\) is a function of neighbouring factor nodes \(f_{m'}\), where \(m' \in N(n) \setminus m\) and \(N(n) \setminus m\) are all neighbouring factor nodes excluding \(f_m\). It reflects the current probability distribution (belief) about \(x_n\) given evidences from all its neighbours. The message has the following element-wise product form:

\[
\nu_{n \rightarrow m}(x_n) = \prod_{m' \in N(n) \setminus m} \mu_{m' \rightarrow n}(x_n). \quad (3.24)
\]

Given the structures of the messages, each message can be computed recursively in terms of other messages in the graph, thus allowing for an efficient algorithm to calculate all the messages. The algorithm begins by viewing a variable node \(x_n\) as the root of the tree-structured graph and initiating messages at the leaves of the graph towards the roots. Once the root node has received messages from all of its neighbours, messages are passed outwards from the root all the way to the leaves, thereby forming a two-pass message passing algorithm. Leaf nodes can either be a factor or variable node, and their initial message has the form

\[
\mu_{m \rightarrow n}(x_n) = f(x_n) \quad (3.25)
\]

or

\[
\nu_{n \rightarrow m}(x_n) = 1, \quad (3.26)
\]

respectively.

At the conclusion of this two-pass algorithm, the current belief about any variable \(x_n\) can be computed by fusing incoming factor-to-variable messages, illustrated diagram-
matically in Fig. 3.7(c) given as

$$P(x_n) = \prod_{m \in N(n)} \mu_{m \rightarrow n}(x_n).$$  (3.27)

**Max-product Algorithm**

Also referred as the max-sum algorithm, the max-product algorithm can be derived by modifying the sum-product algorithm. It is used to calculate the most likely estimation of the hidden variables in a model. More formally, the max-product algorithm is an efficient algorithm for finding the values of hidden variables $X = \{x_1, \ldots, x_N\}$ that maximize the joint distribution $P(X)$. The name max-sum steams from the fact that calculations are generally performed in the logarithmic domain to avoid numeric issues, which converts all products to sums. Here, we will derive the algorithm in the logarithmic domain to produce the max-sum algorithm. Message updates for the max-product algorithm can be obtained by taking the exponential of the derived messages.

The derivation of the algorithm makes use of the following distributive law for the max operator:

$$\max(a + b, a + c) = a + \max(b, c).$$

The distributive law allows the exchange of product with maximization operations in the MAP estimation of the joint distribution in the logarithmic domain with respect to the hidden variables, given as

$$\ln P(X^{\text{max}}) = \max_X \ln P(X),$$

$$= \max_{x_1, \ldots, x_N} \ln P(x_1, \ldots, x_N),$$

$$= \max_{x_1, \ldots, x_N} [\ln P(x_1) + \ln P(x_2) + \ldots + \ln P(x_N)],$$

$$= \max_{x_1} \ln P(x_1) + \max_{x_2} \ln P(x_2) + \ldots + \max_{x_N} \ln P(x_N)$$

where $X^{\text{max}}$ is the optimal setting of $X$ that maximizes the joint probability distribution, i.e., set

$$X^{\text{max}} = \arg \max_X P(X) \equiv \arg \max_X \ln P(X).$$
The above shows that the calculation of the MAP estimate can be factorized similarly to the marginal distribution calculations in the sum-product algorithm. Hence, the messages in the max-sum algorithm are derived in like manner to their counterparts in the sum-product algorithm. Moreover, the message passing procedure in the max-sum algorithm is the same as in the sum-product algorithm except for the messages themselves, i.e., the messages are passed from the leaves to the root (arbitrary any node) and back to the leaves. Based on the messages derived for the sum-product algorithm given in Eq. 3.23 and Eq. 3.24, the messages for the max-sum algorithm can be written down simply by replacing sum with max operators and changing products with sums of logarithms, to give

\[
\mu_{m\rightarrow n}(x_n) = \max_{x_{N(m)}\setminus n} \left[ \ln f_m(x_{N(m)}) + \sum_{n'\in N(m)\setminus n} \nu_{n'\rightarrow m}(x_n) \right], \tag{3.28}
\]

\[
\nu_{n\rightarrow m}(x_n) = \sum_{m'\in N(n)\setminus m} \mu_{m'\rightarrow n}(x_n). \tag{3.29}
\]

The initial messages send from the leaf nodes are obtained by analogy with Eq. 3.25 and Eq. 3.26 and are given by

\[
\mu_{m\rightarrow n}(x_n) = \ln f(x_n), \tag{3.30}
\]

\[
\nu_{n\rightarrow m}(x_n) = 0. \tag{3.31}
\]

After the algorithm is completed, the maximum joint probability distribution for any variable \(x_n\) can be computed using

\[
P(x_n^{\text{max}}) = \max_{x_n} \left[ \sum_{m\in N(n)} \mu_{m\rightarrow n}(x_n) \right]. \tag{3.32}
\]

The process of evaluating the maximum joint probability distribution also gives the most
probable value of $x_n^{\text{max}}$, defined by

$$x_n^{\text{max}} = \arg \max_{x_n} \left[ \sum_{m \in N(n)} \mu_{m \rightarrow n}(x_n) \right].$$

(3.33)

**Loopy Belief Propagation**

As we have shown, exact inference is possible using the sum-product and max product algorithms when a model can be represented in a tree-structured graph without loops. However, an efficient approximate inference method is needed when graphs have loops. Here, we present the loopy belief propagation [163], which builds directly on the sum-product and the max-product algorithm. The idea is simply to apply the sum-product algorithm (or max-product algorithm for MAP estimation) repeatedly on a graph until convergence. This is possible because the message passing rules for the sum-product algorithm are purely local, which means each message uses only messages passed from its immediate neighbours. Since there are cycles in the graph, information can flow many times around the graph, and convergence is not guaranteed. Nevertheless, in practice, it has been applied successfully to applications in medical diagnostics [164], phase-wrapping [165] and handwritten digit recognition [166] to name a few.

There are two components to consider when applying the loopy belief propagation algorithm: the convergence condition and the message passing schedule. Since convergence is not guaranteed, the loopy belief propagation algorithm iterates until convergence or until a predefined number of iterations if convergence is not met. The convergence criterion or a maximum number of iterations to iterate are model and application specific. The second important component to the loopy belief propagation algorithm is the message passing schedule, which there are many. For example, a serial schedule passes one message at a time. Alternatively, a flooding schedule simultaneously passes a message across every link at each time step. The choice of a scheduling method is dependent on the graph structure and the data being analyzed.
3.2.8 Example: Affinity Propagation (AP)

In this section, we demonstrate how to apply loopy belief propagation on a complex clustering model represented as a factor graph with loops. The algorithm, Affinity Propagation (AP) \[166\], is recently introduced to tackle the problem of EBC (similar to \(k\)-medoids clustering (Sec. 3.2.3)). Two of its main benefits are: 1) it simultaneously considers each data point as a possible exemplar and 2) it automatically detects the optimal number of clusters given the model parameters, exemplar preferences. The graphical model representation of AP as a factor graph is shown in Fig. 3.8. This generalized representation of AP allows for easy conversion to solve clustering and facility location problems, and to develop extensions to AP. A more concise representation can be found in \[167\].

![Graphical model for AP drawn as a factor graph](image)
To start, we are given $N$ data points, the similarities $s_{ij}$ between data points $i$ and $j$ and a set of exemplar preferences $\{c_j\}$, where $c_j$ is the preference for choosing point $j$ as the exemplar. The goal is to find a subset of data points as exemplars and assign each non-exemplar data point to exactly one of the exemplars which maximize the exemplar preferences and total similarities between data points and their exemplars. Let $\{h_{ij}\}$ be a set of $N^2$ binary hidden variables, where $i = 1, \ldots, N$ and $j = 1, \ldots, N$ and $h_{ij} = 1$ indicates data point $i$ has chosen data point $j$ as its exemplar. Furthermore, let $\{e_j\}$ be a set of $N$ binary hidden variables, where $j = 1, \ldots, N$ and $e_j = 1$ indicates data point $j$ is chosen as exemplar. The function definitions, specified in the log-domain, are

$$C_j(e_j) = c_j e_j$$  \hspace{1cm} (3.34)$$

$$S_{ij}(h_{ij}) = s_{ij} h_{ij}$$  \hspace{1cm} (3.35)$$

$$I_i(h_i) = \begin{cases} 0 & \text{if } \sum_j h_{ij} = 1, \\ -\infty & \text{otherwise.} \end{cases}$$  \hspace{1cm} (3.36)$$

$$E_j(h_j, e_j) = \begin{cases} 0 & \text{if } e_j = h_{jj}, e_j \geq h_{ij} \forall i, \\ -\infty & \text{otherwise,} \end{cases}$$  \hspace{1cm} (3.37)$$

where the shorthand notations $h_{ij} = h_{1j}, \ldots, h_{Nj}$ and $h_i = h_{i1}, \ldots, h_{iN}$ are used. Intuitively, Eq. 3.34 outputs the exemplar preference for $j$ only if $j$ is an exemplar; Eq. 3.35 outputs the similarity between $i$ and $j$ only if $j$ is chosen as $i$’s exemplar; Eq. 3.36 ensures $i$ chooses only one exemplar; and Eq. 3.37 ensures if $j$ is picked as an exemplar then $j$ picks itself as its exemplar ($h_{jj} = 1$ since $\sum_j h_{ij} = 1$ from Eq. 3.36), where $e_j = h_{jj} = 1$ indicates $j$ is an exemplar.

The goal is therefore to maximize the objective function with respect to $\{h_{ij}\}$ and $\{e_j\}$, given as

$$\mathcal{F}(\{h_{ij}\}, \{e_j\}) = \sum_{i,j} S_{ij}(h_{ij}) + \sum_j C_j(e_j) + \sum_i I_i(h_i) + \sum_j E_j(h_j, e_j).$$  \hspace{1cm} (3.38)$$
The free parameters that control the model complexity are the exemplar preferences \( c_j \)'s, which can be seen as control knobs for the number of clusters to be discovered.

The MAP solution to the problem is obtained by running loopy belief propagation with the max-product algorithm (Sec. 3.2.7). The message updates in the algorithm can be simplified to the following two messages, termed availability (\( \alpha \)) and responsibility (\( \rho \)), that are calculated iteratively until convergence:

\[
\alpha_{ij} = \begin{cases} 
  c_j + \sum_{k \neq j} \max(0, \rho_{kj}) & i = j, \\
  \min[0, c_j + \rho_{jj} + \sum_{k \notin \{i,j\}} \max(0, \rho_{kj})] & i \neq j,
\end{cases}
\]

(3.39)

and

\[
\rho_{ij} = s_{ij} - \max_{k \neq j}(\alpha_{ik} + s_{ik}).
\]

(3.40)

Next, we show the results of AP applied to the clustering problem on the Old Faithful dataset used throughout this chapter. Fig. 3.9 plots the results of the algorithm using negative Euclidean distances between data points as similarities and various settings of \( c_j \). Here, the exemplars are displayed as "*" and each cluster is represented by a different colour. When compared to the other algorithms, \( k \)-means clustering, \( k \)-medoids clustering and MOG, AP finds almost the same cluster assignment for this dataset when the parameter setting yields a result with two clusters. In addition, AP finds the same pair of exemplars as did \( k \)-medoids clustering in those same experiments. Since we are adjusting \( c_j \)'s instead of \( K \) directly, there can be multiple settings of \( c_j \)'s that produce the same number of clusters, as shown in the plots.

Using the Old Faithful dataset, we have shown how all four algorithms work, and that they perform well on this simple dataset. In addition, we have highlighted that MOG and AP can model complex problems with the use of hidden variables, whereas their counterparts, \( k \)-means and \( k \)-medoids clustering, cannot. Lastly, we have illustrated that AP can automatically detect the optimal number of clusters given the model parameters \( c_j \)'s. Whereas MOG, \( k \)-means and \( k \)-medoids clustering require the number of clusters
Chapter 3. Background on Machine Learning Concepts

Figure 3.9: Illustration showing the results of applying AP to cluster the re-scaled Old Faithful dataset used by other clustering algorithms discussed in this chapter. The similarity measures used are the negative Euclidean distances between data points. Various settings of the exemplar preferences $c_j$ are used to demonstrate their effect. The exemplars are displayed as ‘*’ and different colour represents different cluster. Notice that multiple settings of $c_j$ can yield the same number of clusters; both $c_j = -40$ and $c_j = -50$ produce result with two clusters.
$K$ to be defined. For these three algorithms, we will need to perform model selection to find the optimal setting for $K$, which we will discuss next.

### 3.2.9 Model Complexity and Model Selection

In three of the four clustering algorithms above, $k$-means clustering (Sec. 3.2.2), $k$-medoids clustering (Sec. 3.2.3) and MOG (Sec. 3.2.6), the number of clusters $K$ controls the number of free parameters in the model and thereby governs the model complexity [130]. When $K$ is set to a large number, much greater than the underlying optimal number, the model can adjust to very specific random features of the training data that are not reproducible and likely have no relation to the underlying causes that separate the data, a phenomenon referred to as over-fitting. The goal of a good generative model is to perform well on new unseen data. However, in an over-fitted model, the performance decreases on the test dataset, while the model gets more complex and the likelihood increases on the training dataset. This is because over-fitted models begin to ‘memorize’ the features of the training dataset and become less generalizable. As an example, we set $K = N$, where $N$ is the number of training data points, the optimal solution would assign each data point as a cluster by itself. Aside from having numerical issues like singularity with the variances, the model does not produce any meaningful clusters and would perform poorly when given new input data.

The problem of over-fitting is nonexistent if unlimited training data are available (assuming a reasonable $K$ is used), since the true posterior distribution can be determined precisely. However, in many real-life problems, the available training data is quite limited and cannot thoroughly represent the entire feature space. Various model selection methods have been developed to address the issue of identifying the best parameter settings to achieve a reasonable model complexity.

One of the most common approaches to model selection is to train different models (or different settings of the complexity parameters) on a subset of the available data (training
Chapter 3. Background on Machine Learning Concepts

dataset), evaluate the learned models on the rest of the data (generally referred to as a validation dataset), and choose the model with the best result. A problem exists when the validation dataset is small, which will give a relatively noisy estimate of predictive performance. This problem can be overcome by using a process called cross-validation, in which, for each of $S$ runs of the algorithm, a different proportion $(S - 1)/S$ of the available data is used as the training data while the rest is used as validation data for performance assessment. In this type of iterated learning technique, over-fitting to the validation data can occur, so it is common to reserve a third independent test dataset to evaluate the performance of the selected model. A special case of cross-validation, the leave-one-out technique sets $S$ to be equal to the number of training data points, i.e., for each training run one data point is used as the validation set. The major drawback to this model selection approach is the enormous computational time required since the number of training runs grows by a factor of $S$; and this problem is worse when the model has multiple complexity parameters, which could require a number of training runs that is exponential to the number of complexity parameters \[130\].

Alternatively, the selection of $K$ can be done implicitly without adjusting $K$ directly. We have already seen an example of this approach in AP (Sec. 3.2.8). There, the model selection parameters that are varied are the exemplar preferences and $K$ is learned as a result. Another indirect method, which does not require the need for a validation dataset, is to use a Bayesian approach. We will give an example of an such an approach later in Sec. 3.2.10. The Bayesian view of model selection is to use probabilities to represent uncertainties in the choice of the model. Over-fitting can be avoided by marginalizing over model parameters instead of making point estimates of their values, like in non-Bayesian methods such as ML and MAP. This allows all the data to be used for training and eliminates the time-consuming step of training multiple times associated with cross-validation. Also, multiple complexity parameters can be determined simultaneously during the training process. Like other Bayesian learning methods, the key
is to apply a prior probability to the model. The prior represents preferences we have for different models and its choice greatly affects the quality of the final solution as an improper prior can make the resulting equations ill-formed. Therefore, finding the right prior remains a major challenge for Bayesian approaches to model selection. In many real-life applications finding the right prior can be more difficult than coming up with the right model.

For mixture models, the split and merge method [168, 169, 170] offers a non-Bayesian, elegant and effective way to perform model selection in an implicit manner. The idea is to adjust the number of mixture components by iteratively pruning, splitting and merging the components in a mixture model that monotonically improves a cost function, and stops when no improvement can be made. Depending on the problem and the available data, a cost function is chosen such that it can objectively compare the fitness of different models. It is common for the cost function to contain a penalty term $\varphi K$ (or $\varphi P$ where $P$ is the number of parameters), where $\varphi$ is the weight of the penalty; this method favours models with smaller $K$ (or $P$). However, choosing $\varphi$ can be as difficult as choosing $K$. No matter what the cost function is, at the end of each iteration the model is guaranteed to be better in terms of the cost function than the model in the previous iteration. In addition to a cost function, the criteria used to decide which mixture components to prune away, split apart and merge together are also important. There are two main advantages of this type of model selection for mixture models. First, at each iteration the model leverages the result from previous iterations. As a result, it reduces the amount of total computations needed when compared to cross-validation or other brute force methods. Second, more mixture components can be automatically assigned to areas where they are needed and less at others.

A limitation of $k$-means clustering, $k$-medoids clustering and mixture model using the EM algorithm is that clusters (or components) cannot move across or leapfrog over other clusters. Therefore, for instance, if $k$-means clustering is used in Euclidean space, and
an area in the Euclidean space is not initialized with enough clusters, then the algorithm
cannot correct for this and will result in a globally sub-optimal solution. The split and
merge algorithm can fix this by merging and pruning away clusters where fewer clusters
are needed, and splitting clusters where more clusters are required. This feature also
avoids the random restarts of learning algorithms that are commonly used to avoid local
optimum problems, which the cross-validation model selection method suffers from.

3.2.10 Automatic Model Selection with Nonparametric Bayesian
Models

Model selection in the case of clustering focuses on selecting the number of clusters $K$
which should be used to model the data. We have discussed above a number of methods
that address this issue. Nonparametric Bayesian models [171] provide a different approach
to the problem of model selection. Instead of comparing models that vary in complexity,
i.e., models with different $K$, the nonparametric Bayesian approach is to fit a single
model that can adapt its complexity to the data. Moreover, this approach allows the
complexity to grow as more data are observed. As a result, the use of nonparametric
Bayesian models has the benefits of automatic model selection and not requiring any
manual parameterization.

Examples of models that use this approach are infinite mixture models (IMMs) [172]
[173] (also known as Dirichlet process mixture), which are similar to ‘finite’ mixture models
like MOG (Sec. 3.2.6) but without a bound on the number of mixture components that
can be considered. The main idea behind IMMs is that they estimate the number of
clusters needed to model the observed data and allow future data to be assigned to
previously unseen clusters. Therefore, unlike $k$-means, $k$-medoids and MOG, IMMs do
not need to specify $K$ in advance of analyzing the data. As the name implies, an IMM
assumes there is an infinite number of clusters and specifies a prior on the number of
expected clusters in such a way that favours assigning data to a small number of them.
A prior on an uncertain quality $h$ (e.g., a hidden variable or model parameter) is a probability distribution representing one’s belief about the uncertainty about $h$ before the data is observed. For example, if we were to guess the gender of a new baby without having seen him/her, we might guess that there is a 51% chance that the baby is a boy if we use the current ratio of male to female in the world as our prior.

Although in practice manual parameterization can be avoided, nonparametric Bayesian models are not completely without any parameters as the name might suggest. Nonparametric Bayesian models differ from parametric models by placing a prior on model parameters to account for uncertainties and inferring the shape of the prior instead of taking a point estimate of the parameters. Nonparametric models that we are focused on, such as IMMs, place hyperpriors on the priors’ parameters. Hyperpriors are priors placed on parameters of another prior distribution to account for their parameters’ uncertainties. Although not fully Bayesian, it is a common practice to estimate the hyperpriors’ parameters using the input data instead of setting them through means such as model selection methods discussed above. Therefore, this approach of using two layers of priors avoids the need for any manual parameterizations since all parameters are either estimated using the input data (hyperpriors’ parameters) or inferred (all other parameters).

In a nonparametric Bayesian model, each model parameter is given a prior. The prior used in IMMs to generate the parameters of each mixture component is a Dirichlet process (DP) \[174, 175\]. A DP is defined as a distribution over distributions, where each draw from a DP produces a random distribution and the collection of the marginal of each random distribution drawn are Dirichlet distributed. Intuitively, a DP can be thought of as an infinite-dimensional generalization of the Dirichlet distribution. A Dirichlet distribution is a discrete probability distribution and when used as a prior, can have the effect of a sparse prior, where only a small subset of outcomes has high probability, and all others have a near-zero probability. Therefore, a DP prior is a Dirichlet distribution
that can generate an infinite number of distributions but favours a smaller number of active ones. For an IMM, a DP prior provides the model with the ability to describe the data with an infinite number of clusters while only a small number of them are active.

Depending on the specific form of an IMM, the model can contain additional parameters in which priors are placed on them. The choice of priors depends on the model design and availability of prior knowledge. In cases where no specific subject matter knowledge is known, conjugate priors\[130\] are chosen for computational efficiency and tractability reasons. A prior is conjugate to a likelihood distribution when their posterior, i.e., prior $\times$ likelihood, results in a distribution with the same functional form as the prior. As an example, a Dirichlet distribution is the conjugate prior to the parameters of a multinomial distribution. For an infinite MOG model, a Gaussian distribution is used on the parameters means $\mu$ and an inverse gamma distribution is placed on the parameters (scalar) variances $\Sigma$ as they are the conjugate priors for the mean and variance of a Gaussian distribution, respectively.

Illustration of a Bayesian finite mixture model and an IMM using a DP prior is given in Fig. 3.10(a) and Fig. 3.10(b), respectively. Formally, given a DP prior $DP(\gamma, H_0)$, where $\gamma$ is the concentration parameter and $H_0$ is an arbitrary base distribution, a draw from $DP(\gamma, H_0)$ will return a random distribution $H$ containing values drawn from $H_0$. Different draws from a $DP(\gamma, H_0)$ can produce the same $H$, and the frequency of repeats are determined by $\gamma$, where smaller values of $\gamma$ favours more repeats (i.e., equivalent to having fewer number of clusters). The model parameters $\theta_n$ for the $n$-th input data $x_n$ is generated from $H$. Lastly, $x_n$ is drawn from a function $F$ parameterized by $\theta_n$. The model, shown in Fig. 3.10(b), can be described as follows:

$$P(H|\gamma, H_0) = DP(\gamma, H_0),\quad (3.41)$$

$$P(\theta_n|H) = H,\quad (3.42)$$

$$P(x_n|\theta_n) = F(\theta_n).\quad (3.43)$$
Chapter 3. Background on Machine Learning Concepts

Figure 3.10: Bayesian networks of a Bayesian finite MOG, an IMM using a DP prior and simplified IMM with a DP prior. (a) shows a Bayesian network for a general Bayesian finite MOG. Notice the resemblance to the MOG model depicted in Fig. 3.5 on p.61, where the hyperparameter $\gamma$, the concentration parameter to the Dirichlet prior on parameter $\alpha_k$, and prior $H_0$ on the model parameters $\theta_k$ are added, and $\theta_k$ is used in place of parameters $\mu_k$ and $\Sigma_k$. The plate notation (box) indicates that there are $K$ copies of $\alpha_k$ and $\theta_k$, one for each cluster, and there are $N$ copies of $c_n$ and $x_n$, one for each input data. (b) represents an IMM with a DP prior, where the prior (or base distribution) $H_0$ and $\gamma$ are the parameters to and $H$ is the output distribution returned from the DP prior. In this model, though inefficient, there is a copy of $\theta_n$ for each data point. When conjugate priors are used, the IMM can be simplified in (c) by integrating out $H$ and reintroducing cluster assignments $c_n$. As a result, we can store the model parameters in $\theta_k$ that are shared by all data for each of the infinite number of clusters. The latter model is equivalent to using a Chinese Restaurant Process (CPR) as its prior, an alternative representation to DP discussed in Sec. 3.2.10.
In the case of an infinite MOG model, $H_0$ is the conjugate prior, normal inverse gamma, $\theta_n$ represents the parameters means $\mu$ and variances $\Sigma$, and $F(\theta_n) = \mathcal{N}(\mu, \Sigma)$ is the normal distribution parameterized by $\mu$ and $\Sigma$. When conjugate priors are used, the model can be simplified by integrating out $H$ and sampling each $\theta_n$ directly from $H_0$, conditioned on all other parameters. Further enhancement can be made by reintroducing cluster assignments $c_n$ and storing model parameters in $\theta_k$ that are shared by all data in a cluster $k$ instead of one for each data point. This concise model is shown in Fig. 3.10(c).

**Chinese Restaurant Process (CRP)**

A DP can be better explained in the context of its alternative representation, the Chinese Restaurant Process (CRP). The CRP derives its name from the following metaphor; we start with the simpler case of a restaurant with a fixed number of tables, then expend to a scenario where there is an infinite number of tables. Imagine a Chinese restaurant with $K$ tables, and a sequence of customers enters the restaurant and sits down at one of the tables. The CRP assumes that customers are more likely to sit at an occupied table with likelihood proportional to the number of customers already at the table, i.e., tables with more customers are more preferred by a new customer. The process starts with the first customer entering the restaurant and sitting at the first table. The second customer enters and sits at the first table with the probability $\frac{1 + \frac{1}{K}}{1 + \gamma}$ and at another table with the probability $\frac{\gamma}{1 + \gamma}$, where $\gamma \geq 0$ is the concentration parameter, $\frac{\gamma}{K}$ can be viewed as the initial probability of sitting at a new table and the denominator $1 + \gamma$ is the total number of customers seated thus far plus the sum of initial probabilities for all $K$ tables ($\sum_{1}^{K} \frac{\gamma}{K} = \gamma$). The $n$-th customer will sit at an occupied table with a probability proportional to the number of customers already at that table and at the next unoccupied table with probability proportional to $\gamma$. The probability of the $n$-th customer in a CPR is equivalent to the probability of assigning to a cluster given a DP prior in the case of clustering.
More formally, let $c_n$ be the table assignment for the $n$-th customer. The probability of $c_n$ for a new customer depends on the sitting arrangement of previous customers $c_{1,...,n-1}$, given as

$$P(c_n = k|c_{1,...,n-1}, \gamma, K) = \begin{cases} \frac{o_{-n,k} + \beta}{n-1+\gamma} & \text{if } k \leq K_+ \text{ (i.e., } k \text{ is a previously occupied table)}, \\ \frac{\beta}{n-1+\gamma} & \text{otherwise (i.e., } o_{-n,k} = 0) \end{cases}$$

(3.44)

where $o_{-n,k}$ is the number of customers sitting at table $k$ excluding the $n$-th customer and $K_+$ is the set of occupied tables with $o_{-n,k} > 0$.

To convert the CRP from having a finite to an infinite number of tables, imagine we consider the limit as $K \to \infty$ in Eq. [3.44]. For the $n$-th customer, the probability of sitting at an occupied table $k \in K_+$, i.e., $o_{-n,k} > 0$, will then be

$$P(c_n \in K_+|c_{1,...,n-1}, \gamma) = \lim_{K \to \infty} \frac{o_{-n,k} + \beta}{n-1+\gamma} = \frac{o_{-n,k}}{n-1+\gamma}.$$ 

For an unoccupied table $k \in K_-$, i.e., $o_{-n,k} = 0$, as $K \to \infty$ the probability of sitting at that table approaches 0; however, at the same time, the number of unoccupied tables $K_- = K - K_+$ reaches infinity. Therefore, the probability of sitting at any one of the unoccupied tables is the sum over all individual unoccupied table, shown as

$$P(c_n \in K_-|c_{1,...,n-1}, \gamma) = \lim_{K \to \infty} \sum_{K_-} \frac{\beta}{n-1+\gamma} = \frac{\gamma}{n-1+\gamma} \lim_{K \to \infty} \sum_{K_-} \frac{1}{K} = \frac{\gamma}{n-1+\gamma} \lim_{K \to \infty} \frac{K}{K} = \frac{\gamma}{n-1+\gamma} \lim_{K \to \infty} \frac{K-K_+}{K} = \frac{\gamma}{n-1+\gamma}.$$

Putting the two above equations together, the probability of sitting at table $k$ for the $n$-th customer is given as

$$P(c_n = k|c_{1,...,n-1}, \gamma) = \begin{cases} \frac{o_{-n,k}}{n-1+\gamma} & \text{if } k \in K_+ \text{ (i.e., } k \text{ is a previously occupied table)} \\ \frac{\gamma}{n-1+\gamma} & \text{otherwise (i.e., } o_{-n,k} = 0) \end{cases}$$

(3.45)
which implies that the probability of sitting at an occupied table is proportional to
the number of customers at that table already, while the probability of sitting at an
unoccupied table is proportion to $\gamma$. The CRP is therefore the a stochastic process
whose probability distribution is given in Eq. 3.45.

Given Eq. 3.45, we can interpret the parameter $\gamma$ as a pseudocount of customers
sitting at each table (note that $\gamma$ does not have to be an integer), where a larger value
will produce more occupied tables with fewer customers per table. To see the effect of
different settings of $\gamma$, let’s consider an example when the second customer arrives at a
restaurant with an infinite number of tables. For $\gamma = 2$, the second customer will have
the chance $\frac{1}{(2-1+2)} = 0.33$ of sitting at the first table occupied by the first customer and
$\frac{2}{2-1+2} = 0.67$ of sitting at a new table; this setting encourages the customer to sit at an
unoccupied table. Whereas, when $\gamma = 1$, the same second customer will have an equal
chance (i.e., $\frac{1}{2-1+1} = 0.5$) of sitting at the first or an unoccupied table. The influence
of $\gamma$ will be less as more customers enter the restaurant. In a nonparametric Bayesian
setting, the parameter $\gamma$ is treated as an unknown and a prior is put over it so that its
value is inferred.

The above description of the CPR assumes customers have no preference on sitting
at a table once assigned. More generally, the probability of sitting at table $k$ for the
$n$-th customer can be modified to include the likelihood of sitting at each table. In the
example of infinite MOG given above in Sec. 3.2.10, where $x_n$’s are the observations,
Eq. 3.45 can alter to the following general form:

$$
P(c_n = k|c_1,\ldots,n-1, \gamma) = \begin{cases} 
    b\frac{o_{n,k}}{n-1+\gamma} F(x_n, \theta) & \text{if } k \in K_+ \text{ (i.e., } o_{n,k} > 0) \\
    b\frac{\gamma}{n-1+\gamma} \int F(x_n, \theta) dH_0(\theta) & \text{otherwise (i.e., } n_{-n,k} = 0),
\end{cases}
$$

(3.46)

where $\theta$ are the model parameters; $F(x_n, \theta)$ is the likelihood function, e.g., $\mathcal{N}(x_n; \theta)$;
$H_0$ is the prior on $\theta$; and $b$ is the appropriate normalizing constant so the probabilities
sum to one. When $H_0$ is a conjugate prior for $F$, the integral $\int F(x_n, \theta) dH_0(\theta)$ may be
analytically computed.

An IMM with a DP prior can be equivalently converted to an IMM with a CPR prior, where Bayesian network representations of each are shown in Fig. 3.10(b) and Fig. 3.10(c) on p.83 respectively. As discussed earlier, when conjugate priors are used, the model can be simplified by integrating out $H$ and sampling $\theta_k$ for each cluster directly from the prior distribution $H_0$, conditioned on all other parameters. In this formation of the model, we reintroduce cluster assignments $c_n$ and store the parameters that are shared by all data in each cluster in $\theta_k$. For each data point, the probability of $c_n$ is given by Eq. 3.46 and if the assignment is to a new cluster $k$ then a new value for $\theta_k$ is drawn from $H_0$.

The next step after defining the model is to perform inference. For models such as IMMs, an efficient method is to examine the posterior distribution of the hidden variables given the data. However, for many complex models, including the ones used here in this section and in our second PTM refinement algorithm, the posterior distribution cannot be fully characterized analytically. To overcome this, various approximate inference algorithms like variational methods, EP [156] and sampling approaches have been developed. Although variational methods, such as mean field variational inference [153, 176, 177, 178], have been shown to converge faster than sampling approaches, they can produce solutions that are worse and are sensitive to initialization, i.e., can be trapped in local optimal solutions. Similarly, EP has is faster but can produce less accurate solutions than sampling methods. For these reasons, in this thesis we decided to focus on approximate inference using sampling approaches on the problem of clustering.

3.2.11 Inference by Sampling

The idea behind inference by sampling is that the posterior can be cleverly sampled to collect instances of probable parameter and variable settings. By collecting enough samples, where the likelihood of each sample being drawn is equal to its probability under
the posterior, the posterior distribution can be estimated. Inference can then be made by taking the expected value of parameters and hidden variables across samples. This averaging across samples approach provides additional statistical information that is not available with point estimate approaches like MAP estimation. An MAP estimation takes the setting that yields the maximum probability under the posterior distribution.

Inference by sampling can be applied to any inference problems. It has been a cornerstone of Bayesian statistics and has become increasingly popular with applications in computational biology [179, 180, 181, 182]. Here, we will focus on a special class of sampling methods called Markov chain Monte Carlo (MCMC) [172, 172, 183]. Specifically, we will discuss two variants of MCMC: Gibbs sampling [184] and split-merge Metropolis-Hastings sampling [185].

**Inference with Gibbs Sampling**

One of the most commonly used MCMC sampling methods for mixture models is the Gibbs sampling [130, 184]. Gibbs sampling is applicable when the joint distribution is not known explicitly or is difficult to draw from directly, but the conditional distribution of each variable is known and is easy (or at least, easier) to sample from. Intuitively, Gibbs sampling starts with an initial setting for hidden variables (model parameters are treated as hidden variables) and then, for each hidden variable, draws a new value based on its conditional distribution conditioned on the current setting of all other variables in the model. An iteration of Gibbs sampling is completed when a new value is drawn for each variable in the model. This sampling procedure is repeated until enough samples are collected to properly represent the posterior distribution. Theoretically, an infinite number of samples is needed but in practice, only a finite number of samples is collected. Similar to other MCMC sampling methods, this strategy yields more samples in high probability regions of the posterior. This means that individual settings of variables that contribute to a higher posterior probability will appear more often.
The Gibbs sampling method has a number of limitations and drawbacks. One requirement for using the Gibbs sampling is that the conditional distribution of each variable conditioned on all other variables must be able to be written in closed form and sampled from. Additionally, although the implementation of Gibbs sampling is straightforward, the algorithm can be slow to converge and can mix poorly [185]. This is because it samples one variable at a time, which equates to exploring the posterior distribution incrementally with small steps. In the context of mixture models, the algorithm updates the clustering assignment of one data point at a time and is unable to move a group of data points to a new cluster. Therefore, when two or more mixture components have similar parameters, the Gibbs sampling method may become trapped in local mode (a local peak in the posterior distribution) that corresponds to an incorrect clustering of data points.

Inference with Split-merge Metropolis-Hastings Algorithm

The split-merge Metropolis-Hastings sampling algorithm can be used to address the aforementioned problems with Gibbs sampling. Intuitively, the split-merge sampling method works similar to the split and merge model selection method (see Sec. 3.2.9). The split-merge sampling algorithm iteratively splits or merges candidate clusters and accepts such moves if they meet certain performance criteria.

The split-merge sampling algorithm is a special case of Metropolis-Hastings algorithms [186], another type of MCMC sampling methods. The idea behind Metropolis-Hastings algorithms is to sample a new set of hidden variables $c'$ from a simpler proposal density $Q(c'|c_t)$ that is proportional to the posterior distribution of interest $P(c_t)$ instead of $P(c')$ directly, where $c'$ is the set of cluster assignments at iteration $t$. $Q(c'|c_t)$ can be seen as a transition probability distribution that describes the likelihood of going from $c_t$ to $c'$. The proposal will be accepted (i.e., $c_{t+1} = c'$) with probability

$$a(c', c_t) = \min \left[ 1, \frac{P(c')Q(c'|c_t)}{P(c_t)Q(c'|c')} \right].$$

(3.47)
If the proposal is rejected, then the variable setting remains the same (i.e., \( c^{t+1} = c^t \)). From Eq. 3.47, a new proposal is more likely to be accepted if the new setting is more probable under the posterior than the previous; this is always true when \( Q \) is symmetric, i.e., \( Q(c'|ct) = Q(c'|c') \). For a new sample \( c' \) that is less probable than \( c^t \) under the posterior, the algorithm accepts it with the probability \( a(c',c^t) \). In contrast, Metropolis-Hastings algorithms can make non-incremental (major) changes with the parameter setting while Gibbs sampling performs incremental (minor) steps. The key to Metropolis-Hastings algorithms is the decision on an appropriate proposal density \( Q \), which can be difficult to find. The split-merge Metropolis-Hastings sampling algorithm provides an elegant solution to this.

For the split-merge Metropolis-Hastings sampling algorithm, a newly proposed state for the cluster assignments is generated by randomly selecting two data points, \( i \) and \( j \) with corresponding cluster assignments \( c_i \) and \( c_j \). Then, it either a) splits the cluster \( i \) and \( j \) belong to if \( c_i = c_j \) (the algorithm heuristically ensures that \( i \) and \( j \) are re-assigned to different clusters) or b) merges the clusters indexed by \( c_i \) and \( c_j \) if \( c_i \neq c_j \). In this split-merge step, only data points belonging to the cluster(s) indexed by \( c_i \) and \( c_j \) are affected (i.e., \( \forall k : c_k = c_j \) or \( c_k = c_j \)) and all other data points in the dataset are not altered. Let \( c^{\text{split}} \) and \( c^{\text{merge}} \) represent the new proposed cluster assignments for splitting or merging the initial cluster(s) indexed by \( c_i \) and \( c_j \), respectively, i.e., \( c' \in \{c^{\text{merge}}, c^{\text{split}}\} \). Then, \( Q(c^{\text{split}}|c^t) \) is the probability of seeing the specific cluster assignment \( c^{\text{split}} \) amongst all possible ways to split the involved data points into two clusters, given as

\[
Q(c^{\text{split}}|c^t) = \left( \frac{1}{2} \right)^{n_{c_i}^{\text{split}} + n_{c_j}^{\text{split}}} - 2,
\]

(3.48)

where \( n_{c_i}^{\text{split}} \) (\( n_{c_j}^{\text{split}} \)) is the number of data points originally assigned to cluster \( c_i \) (\( c_j \)). It is easy to see that \( Q(c'|c^{\text{merge}}) \) is equivalent to \( Q(c^{\text{split}}|c^t) \). The probability of proposing a merge move for two separate clusters is

\[
Q(c'|c^{\text{merge}}) = 1,
\]

(3.49)
since there is only one way to merge two clusters together. Note that $Q(c' | c^{\text{split}})$ is equivalent to $Q(c^{\text{merge}} | c^t)$. The new split (merge) move is accepted with a probability given in Eq. 3.47 at which point the new split (merged) state becomes the new state, i.e., $c^{t+1} = c^{\text{split}}$ ($c^{t+1} = c^{\text{merge}}$). If the move is rejected, then the old state prior to the split (merge) is assigned as the new state, i.e., $c^{t+1} = c^t$. The algorithm repeats the split-merge step for a predefined number of iterations.

The base variant of this algorithm calls for a random assignment of data points into the two new clusters when splitting. The authors of the split-merge Metropolis-Hastings sampling algorithm also suggested a variant to this basic variant, called restricted Gibbs sampling split-merge algorithm [185]. In this improved variant, the algorithm mixes in Gibbs sampling within and after each split-merge step. Although the setting is data dependent, the authors recommended, for each iteration of the algorithm, to perform five Gibbs sampling steps on the split clusters, then two or three split-merge steps and one final Gibbs sampling step after split-merge steps are completed. The restricted Gibbs sampling split-merge algorithm has been shown to outperform Gibbs sampling on clustering problems using IMMs [185].

3.2.12 ML Approaches to PTM Refinement and Protein Identification

The two problems that we set out to solve in this thesis, PTM refinement and protein identification, can be addressed using ML approaches. In this chapter, we have outlined various probabilistic model representations and surveyed the different types of ML algorithms used to learn and infer these models. We have discussed further a class of ML algorithm, clustering, that aims to identify the hidden structure within the input data to form meaningful groupings. Through the use of four distinct clustering methods, we have outlined key algorithmic approaches to clustering and highlighted their strengths and weaknesses. Furthermore, we discussed the importance of model selection and pre-
presented two different types: the parametric approach and the nonparametric approach using Bayesian statistics. In our discussion of nonparametric approaches, we highlighted IMMs as examples of models that use the nonparametric model selection approach, introduced inference by sampling and detailed two MCMC sampling methods, the Gibbs sampling and the split-merge Metropolis-Hastings algorithm. The material presented here in this chapter and Ch. 2 forms the basis of our unique approaches to solving the PTM refinement and the protein identification problem through clustering that we will introduce in the following chapters.
Chapter 4

Computational Refinement of PTM Predictions

A post-translational modification (PTM) is a naturally occurring chemical modification of a protein. Many of these modifications, such as phosphorylation, are known to play pivotal roles in the regulation of protein function. Furthermore, PTM perturbations have been linked to diverse diseases like Parkinson’s, Alzheimer’s, diabetes and cancer \[19, 20, 21, 187\]. Therefore, the detection and discovery of PTMs on the genome-wide scale are becoming major research focuses. To discover PTMs on a genome-wide scale, there is a recent surge of interest in analyzing MS/MS data, and several unrestrictive (so-called blind) PTM search methods have been reported. However, these approaches are subject to noise in mass measurements and in the predicted modification site (amino acid position) within peptides, and hence can result in false PTM assignments.

To address these issues, we devised a machine learning algorithm, PTMClust, that can be applied to the output of blind PTM search methods to improve prediction quality of the modification mass and modification sites. PTMClust achieves this by suppressing noise in the data and clustering peptides with the same underlying modification to form PTM groups. PTMClust outperforms two standard clustering algorithms, \(k\)-means
clustering and MOG, on a simulated dataset. Additionally, our algorithm significantly improves sensitivity and specificity when applied to the output of three different blind PTM search engines, SIMS, InsPecT and MODmap, on a dataset consisting of annotated phosphopeptides. For the same phosphopeptide dataset, when compared to another PTM refinement algorithm, PTMFinder, PTMClust markedly outperforms it. We demonstrate that our technique is able to reduce false PTM assignments, improve overall detection coverage, and facilitate novel PTM discovery, including terminus modifications. We report that our technique can find numerous known and unknown PTMs in an analysis of a large-scale yeast MS/MS proteome profiling dataset. Accurately identifying modifications in protein sequences is a critical first step for PTM profiling, and thus our approach may benefit routine proteomics analysis.

In this chapter, we first highlight the challenges with current PTM prediction methods. Subsequently, we illustrate the need for PTM refinement and present our solution to the problem. To begin, we survey existing PTM prediction and state-of-the-art PTM refinement algorithms. Next, we detail our algorithm PTMClust, which includes a graphical representation and description of the model, and the inference and model selection algorithms that are used. There will be two sets of experiments: first, a set of benchmark experiments on both synthetic and real phosphopeptide data to demonstrate that PTMClust outperforms current PTM prediction and PTM refinement algorithms; second, an experiment on a large-scale yeast study which underscores PTMClust’s utility to improve identification of many known PTMs. We will conclude with a summary, discussion of limitations and future research directions.

4.1 Introduction

A simple review of the molecular biology of a cell, including proteins, PTMs and their roles in the cell, is given in Ch. 2. To summarize, proteins are created through a biological
process called protein biosynthesis. Protein biosynthesis begins with transcription and splicing of genes into messenger RNA molecules (mRNA), then followed by translation of mRNA into proteins. At the time of translation, a protein can either be active or inactive, and its subsequent activity can be regulated by post-translational modifications (PTMs). PTMs, which may occur during or after translation, involve an enzymatic addition of a chemical group (e.g., a phosphate) or a larger moiety (e.g., an additional polypeptide such as ubiquitin), onto one or more amino acid side chains. Many PTMs, in particular, phosphorylation on serine (S), threonine (T) or tyrosine (Y), can regulate a protein’s function by influencing the protein’s folding, stability or physical association with other proteins, thereby activating or suppressing it.

Since PTMs have been shown to dynamically influence a wide range of important processes (e.g., catalysis of biochemical reactions, intracellular cell signalling and cell division), mapping of PTMs in a comprehensive genome-wide manner remains a critical outstanding research problem. Although the biological importance of certain PTMs is well established, the diversity and the prevalence of PTMs and their targets remain to be fully elucidated. To this end, a number of experimental approaches have been developed. One recently developed state-of-the-art approach to discover PTMs on a genome-wide scale is to analyze MS/MS data using an unrestricted (or blind) PTM search engine (see Sec. 2.3 for details). The traditional ‘restricted’ search methods limits themselves to finding only unmodified peptide sequences and peptide sequences that contains specific PTMs taken from a small, predetermined list of candidate PTMs, with predefined delta masses or preferred target residues. On the other hand, blind search engines require no such predetermined list; therefore, they do not have the same limitation as restricted search methods. This allows blind PTM search engines to be able to consider a large number of potential PTMs at once, representing both previously known PTMs and new ones.

As discussed in Sec. 2.4, in practice, blind PTM search methods suffer from two major
Figure 4.1: Histograms of inputs to our algorithm (generated by SIMS [8]) for spectra previously determined to be mapped to phosphopeptides [189]. They show that the statistics for modification mass and modified amino acid deviate from the reference, which determined that the PTM (phosphorylation) occurs at \( \sim 80 \) Da and on serine (S) and threonine (T). (a) shows the distribution of the measured modification mass. (b) shows identified amino acids deviate from S and T. (c) shows the distance (in residues) from the identified amino acid to the reference for misplaced modifications and demonstrates that identified modifications are generally only a few residues away from the reference.
sources of error: sequence-dependent uncertainty in the modification position (residue position along the peptide sequence where the modification is deemed to occur) and mass inaccuracy for the modification mass (delta mass between the mass of the peptide sequence and the observed mass, which is attributed to the addition of PTM). The peptide fragmentation process in MS/MS is often incomplete, and the presence of labile PTMs may interfere with this process [85]. These issues combined result in MS/MS spectra missing peaks, which in turn may lead to ambiguous or erroneous PTM predictions. The presences of natural stable isotopes, such as carbon-13, in addition to electronic noise are major contributors to inaccurate mass measurements. This is more prominent in spectra generated from low mass resolution mass spectrometers (e.g., ion trap mass spectrometers), which are still commonly used in today’s mass spectrometry studies.

Fig. 4.1 shows a diagrammatic representation of the search results obtained from applying the blind PTM search engine SIMS [8] to a set of MS/MS spectra previously mapped to phosphopeptides [189]. Enriched for phosphopeptides using a strong cation exchange based method, the spectra from the complex peptide mixture in the original study were analyzed by a restricted PTM search method designed to look for phosphorylation and were validated manually. The same dataset has been used in benchmark experiments in previous PTM studies [8, 39]. As a reminder, phosphorylation is known to occur at ~80 Da and primarily on the amino acid serine (S) and less frequently on threonine (T). Nevertheless, the results show many of the modification masses and modified amino acid sites outputted by SIMS deviate from this reference. A closer look (Fig. 4.1(c)) shows that many of the misplaced modified amino acids are a few residues away from their corresponding reference modification position. In a global-scale PTM survey, these issues can make distinguishing true PTM matches from false detections non-trivial; therefore, identifying bona fide PTMs confidently remains difficult. While these errors can potentially be reduced by technological improvements in instrumentation (e.g., higher mass accuracy mass spectrometers or using alternate fragmentation
we sought to develop an algorithm that can deconvolve errors associated with measuring masses and mapping of modification positions simultaneously to salvage both existing datasets and current experimental platforms.

To this end, we introduce a novel generative probability model (PTMClust) that addresses the aforementioned problems encountered when using blind PTM search engines. It accomplishes a significant boost in PTM prediction accuracy and precision by modelling the hidden relationships between the compositions of amino acids in the peptide sequence, specifically the modification mass, the modification position, and the identity of the modified amino acid. Our algorithm iterates between clustering modified peptides with similar modification to form groups, which we call PTM groups, and finding the most likely modification mass and modification position for each peptide based on the grouping. Our method distinguishes itself from others by modelling modifications at the PTM level instead of at the individual peptide level. This is significant because, as our results show, majority of modified peptides are detected with few instances even in a genome-wide study. Since approaches modelling modifications at the peptide level require each peptide variant to occur numerous times in the dataset, they can perform poorly when modified peptides are detected multiple times. By rigorous benchmarking, we show that a number of learned PTM groups correspond to known PTMs, and many reported modified peptides match to annotated modifications. In addition, our algorithm simultaneously considers PTMs occurring in either the middle or at the terminal ends of a peptide or protein, which provides additional information missed by blind PTM search techniques [8, 39, 78, 80, 81, 82]. To ensure broad applicability, we have designed and optimized PTMClust to analyze PTM data generated from low resolution MS/MS spectra processed by popular blind PTM search engines, such as those generated from ion trap mass spectrometers.
4.2 PTMClust Algorithm

Our proposed algorithm PTMClust consists of a generative model (see Sec. 3.1), which captures the hidden relationship between factors that influence the PTM mapping process, and an algorithm to infer the values of the hidden variables and parameters. PTMClust also includes a background model to account for spurious data. The input to PTMClust is a set of modified peptide predictions obtained using a blind PTM search method (e.g., SIMS or InsPecT [39], described in Sec. 2.4). The input consists of a list of modified peptides with the following attributes: peptide sequence, measured modification mass and estimated position of the modification along the peptide sequence (modification position). The output of PTMClust for each input peptide entry consists of a cluster assignment, corrected modification position and modification mass of its assigned cluster. The identity of the refined modified amino acid for each peptide can be obtained from its peptide sequence and corrected modification position.

A key component of our algorithm is the model selection method that selects the appropriate number of clusters by adjusting the model complexity parameter (‘control knob’) $\alpha^b$. Using the labels of real and decoy peptides, we define rate of detection (RD) as the number of real peptides that are not assigned to the background model, divided by the total number of real peptides. Similarly, we define rate of false detection (RFD) as the number of decoy peptides that are not assigned to the background model, divided by the total number of decoy peptides. A setting for $\alpha^b$ is chosen by weighing the trade-off between the number of decoy peptides allowed and the number of real peptides detected.

A Generative Model for Finding PTM Groups

By accounting for combinatorial interactions between hidden variables that play a role in the protein modification process, our generative probability model aims to describe how each PTM observation is generated. For a given PTM type (PTM group), the observed
modification mass is assumed to be a noisy version of the expected (mean) modification mass, and the modified amino acid is chosen from a distribution over amino acids that may be modified in that PTM type. For example, modifications occur primarily on serine (S) and threonine (T) for phosphorylation. For a given peptide, the true modification position is assumed to be chosen uniformly amongst occurrences of the chosen modified amino acid in the peptide. Finally, the observed modification position is assumed to be a noisy version of the true position. Below, we describe the components of our model: a) the probability of choosing each PTM type, b) the probability of choosing each amino acid to be the modified amino acid given the PTM type, c) the probability of the true modification position given the modified amino acid, and d) the uncertainty in the observed modification mass and modification position. We then introduce an algorithm for learning the model parameters and inferring the hidden variables from the input data. Once the model is learned, we can refine the modification for each input peptide sequence by inferring its most likely PTM group, true modification mass and true modification position.

The structural relationship between the variables described below is described by the Bayesian network depicted in Fig. 4.2. The figure describes the model for one input and is repeated for $N$ inputs, as indicated by the plate notation.

In our model, each input peptide sequence $S_n$, indexed by $n \in \{1, \ldots, N\}$, where $N$ is the number of peptides in the dataset, has a corresponding discrete peptide length $L_n$, an observed modification position $x_n \in \{1, \ldots, L_n\}$, and an observed modification mass $m_n$. We use $S_n(j)$ to denote the identity of the $j$-th amino acid ($i \leq j \leq L_n$) in the $n$-th input sequence. The total number of values $S_n(j)$ can take on is $A = 24$, which includes the 20 naturally-occurring amino acids and four special characters indicating the beginning and end of proteins and peptides. The hidden variable $c_n \in [1, \ldots, K]$ denotes the unknown PTM group for peptide sequence $n$, where $K$ is the number of PTM groups and will be adjusted depending on the desired false detection rate, as describe later. The prior
Figure 4.2: A Bayesian network describing our generative model, using plate notation. The shaded nodes represent observed variables, the unshaded nodes represent hidden variables and the variables outside the plate are model parameters. The model describes how the observed modification mass and modification position are generated. Given the type of PTM (PTM group), we can generate the observed modification mass as a noisy version of the modification mass mean, and select an amino acid to have the modification as the modified amino acid. Given the peptide sequence, we can choose a position along it that matches the modified amino acid as the ‘true’ modification position. We can generate the observed modification position as a noisy version of the ‘true’ modification position. The plate notation indicates there are N copies of the model, one for each input modified peptide sequence with a modification mass and modification position.
probability (mixing coefficient) for each PTM group is given as

\[ P(c_n = k) = \alpha_k, \] (4.1)

where it satisfies the constraints \( \alpha_k \geq 0 \) and \( \sum_{k=1}^{K} \alpha_k = 1 \), and is inferred from the data (see below for details).

The probability that a PTM occurs on amino acid \( i \in \{1, ..., A\} \), given that the PTM group is \( k \), is

\[ P(a_n = i|c_n = k) = \beta_{ki}, \] (4.2)

where the hidden variable \( a_n \) denotes the true (unobserved) modified amino acid and the \( \beta \)'s satisfy the constraints \( \beta_{ki} \geq 0 \) and \( \sum_{i=1}^{A} \beta_{ki} = 1 \); the \( \beta \)'s are inferred from the data (see below for details).

Given the peptide sequence \( S_n \) and the modified amino acid \( a_n \), each occurrence of that amino acid in the peptide sequence has equal probability of being the true (unobserved) modification position \( z_n \). For completeness, our probabilistic model considers the likelihood of cases where an amino acid does not occur in \( S_n \). To do so, the model allows for the event that the true PTM occurs outside of the given peptide sequence indicated by \( z_n = 0 \), so that \( z_n \in \{0, ..., L_n\} \). All other positions in the peptide sequence have zero probabilities of being the true modification position. This can be written as

\[
P(z_n = j|a_n = i, S_n) = \begin{cases} 
\frac{1}{\delta_{ni}+1} & \text{if } S_n(j) = i, j \geq 1, \\
\frac{1}{\delta_{ni}+1} & \text{if } j = 0, \\
0 & \text{otherwise,}
\end{cases} \] (4.3)

where \( \delta_{ni} \) denotes the number of times amino acid \( i \) occurs in sequence \( n \).

We modelled the modification position error \( (x_n - z_n) \) between the observed modification position \( x_n \) and the true modification position \( z_n \) with a discrete probability

\(^{1}z_n = 0 \) is needed to avoid numerical issues since our algorithm considers each amino acid as a possible modification target.
Figure 4.3: Distribution of modified peptides in the yeast proteome dataset with more than three instances. The data is taken from a large-scale yeast proteome study of yeast whole-cell lysate [121], and the peptides are grouped by their peptide sequence identities and modification mass. The distribution exhibits a long-tail curve where majority of the counts are for modified peptides with less than 4 instances. Modified peptides with three instances consist of $>48\%$ of entries.
distribution, given by

\[
P(x_n | z_n = j) = \begin{cases} 
\phi(x_n - j) & \text{if } j > 0, \\
\phi(L_n) & \text{if } j = 0, \\
0 & \text{otherwise,}
\end{cases}
\]  

(4.4)

where the likelihood function \( \phi \) accounts for the modification position error. This likelihood function is shared across all PTM groups and was inferred from our empirical observation of the yeast PTM dataset as follows (see Sec. 4.3.3 for description of the dataset). We grouped the entries in the dataset by their peptide sequence and modification mass, allowing for mass differences of \( \pm 2 \text{ Da} \). Next, we determined the average modification position for each group (rounded to the nearest position) and computed a histogram of the modification position error. In the above assignment, groups with less than three entries were removed. This threshold was chosen so that a reasonable number of points is available to estimate the mean and variance, a large enough dataset (1206 entries) to estimate the likelihood function is provided, and it can act as a crude filter for false modified peptides. The frequency of peptides for each group size, shown in Fig. 4.3, exhibits a heavy-tail curve indicating the majority of modified peptides have low counts. More than 48% of the entries have a group of size exactly three. The resulting likelihood distribution is shown in Fig. 4.4.

Lastly, we accounted for the variation (noise) in the estimated modification mass by assuming that the observed modification mass for each PTM group is normally distributed around the true modification mass. This gives us

\[
P(m_n | c_n = k) = \frac{1}{\sqrt{2\pi\Sigma_k}} \exp \left( -\frac{(m_n - \mu_k)^2}{2\Sigma_k} \right),
\]  

(4.5)

where \( \mu_k \) and \( \Sigma_k \) are the modification mass mean and variance for the \( k \)-th PTM group, respectively, and are inferred from the data.

By combining the structure of the Bayesian network and the conditional distributions
Figure 4.4: Distribution of modification position error used by PTMClust. This empirical distribution is derived using yeast PTM data [121] analyzed with SIMS [8]. A positive (negative) modification position error indicates the observed modification position is towards the C-terminus (N-terminus) of the expected modification position.
described above, we can write the joint distribution as

\[
P(c, a, z, x, m|S, \theta) = \prod_{n=1}^{N} \left( P(c_n|\theta)P(m_n|c_n, \theta)P(a_n|c_n, \theta)P(z_n|a_n, S_n, \theta)P(x_n|z_n, \theta) \right),
\]  

(4.6)

where \(\theta\) represents the model parameters \((\alpha_k, \beta_{ki}, \mu_k\) and \(\Sigma_k)\).

The input data are noisy and may contain false positives and modified peptides that do not fit into proper PTM groups. To account for these spurious data points, we included an additional PTM group (background component) that acts as a garbage collection process (background model). In this background component, we assumed there is no specific relationship between the modification mass and modified amino acid. Formally, the background component has a fixed modification mass mean \(\mu^b\) and variance \(\Sigma^b\) set to be equal to the mean and variance of the data. Additionally, it has a fixed uniform probability over the modified amino acid \(\beta^b_a = \frac{1}{A}, \forall a = 1, ..., A\), and a mixing coefficient \(\alpha^b\), which will be used to adjust model complexity (see below).

**Inference and Learning**

The key step in our algorithm is to infer an optimal setting for hidden variables and learn the model parameters. However, exact inference and learning of the PTMClust model are computationally intractable because of nonlinear relationships between hidden variables and parameters. Instead, our algorithm uses the EM algorithm (Sec. 3.2.5), which alternates between probabilistically filling in the hidden variables \(c_n, a_n\) and \(z_n\) and estimating the parameters \(\alpha_k, \beta_{ki}, \mu_k\), and \(\Sigma_k\). A detailed derivation of the EM algorithm for our model is provided in Appendix B.

In the E-step, the posterior probability for iteration \(t\) and each peptide \(n \in \{1, ..., N\}\) is evaluated using the parameters from iteration \(t - 1\) by conditioning on the observed
variables \( m_n \) and \( x_n \) in (4.6), given as

\[
Q^{(t)}(c_n, a_n, z_n) = P(c_n, a_n, z_n|m_n, x_n, S_n, \theta^{t-1}) = \frac{P(c_n, a_n, z_n|m_n, x_n, S_n, \theta^{t-1})}{\sum_{c_{n}} \sum_{a_{n}} \sum_{z_{n}} P(c_n, a_n, z_n, m_n, x_n|\theta^{t-1})}.
\]

(4.7)

In the M-step, the parameters are re-estimated by maximizing the expected complete log-likelihood using the current posterior probabilities. This is done by taking the partial derivative of the expected complete log-likelihood with respect to each parameter. Lagrangian terms are added to the expected complete log-likelihood to account for the constraints on parameters \( \alpha_k \) and \( \beta_{ki} \). The updates for the parameters are as follows:

\[
\mu_k = \frac{\sum_{n=1}^{N} Q^{(t)}(c_n = k)m_n}{\sum_{n=1}^{N} Q^{(t)}(c_n = k)}, \quad \Sigma_k = \frac{\sum_{n=1}^{N} Q^{(t)}(c_n = k)(m_n - \mu_k)^2}{\sum_{n=1}^{N} Q^{(t)}(c_n = k)},
\]

\[
\alpha_k = \frac{1}{N} \sum_{n=1}^{N} Q^{(t)}(c_n = k), \quad \beta_{ki} = \frac{\sum_{n=1}^{N} Q^{(t)}(c_n = k, a_n = i)}{\sum_{n=1}^{N} Q^{(t)}(c_n = k)}.
\]

(4.8)

At the end of each pair of E- and M-steps, the algorithm calculates the log-likelihood and stops if the difference between the current and previous log-likelihood divided by the current log-likelihood is smaller than \( 10^{-5} \) (this stop criterion is chosen to ensure the EM algorithm converges).

**Recursive Merge Method for Model Selection**

In our model, the only free parameter is the number of PTM groups (mixture components) \( K \). We devised a recursive merge method, similar to split and merge model selection methods\(^2\) discussed in Sec. 3.2.9 to evaluate and identify the optimal setting for \( K \). An optimal setting for \( K \) is chosen such that it achieves a desired false detection rate.

Instead of adjusting \( K \) directly, the algorithm adjusts the mixing coefficient of the background component \( \alpha^b \) to achieve the desired model complexity. \( \alpha^b \) represents the

\(^2\)Our approach only makes use of merge steps.
prior probability that a data point belongs to the background model; a large value for $\alpha^b$ more input data should be assigned to the background model. For each specific setting of $\alpha^b$, i.e., a control knob setting, our method infers the hidden variables, parameter settings and $K$, as described above. Using maximum likelihood estimation and a step size of 0.01 for $\alpha^b$, as $\alpha^b$ increases, more and more of the loosely clustered peptide sequences are redistributed to other components, including the background component, and the number of non-background components decreases. This is accomplished by pruning away ‘empty’ components, where we define a component to be empty when it has less than or equal to one peptide sequence assigned to it. In effect, as $\alpha^b$ increases, the non-background components are slowly merging with each other and with the background component until the non-background components are empty and pruned away, which decreases the model complexity. Our algorithm starts with a large value for $K$ and a small value for $\alpha^b$ (0.01), and slowly merges the non-background components and the background component, pruning away any empty clusters, by increasing $\alpha^b$ each time. In essence, we learn $M$ models, where $M$ is the number of different $\alpha^b$ settings. We choose a single model (i.e., a specific setting for $\alpha^b$) by analyzing the results from our model selection method using the RD and RFD measures. The choice of which model to use depends on the desired RD and RFD as any such choice is dataset dependent.

**Synthetic PTM Data Generation**

To compare PTMClust against standard clustering algorithms for finding correct groupings of modifications, we generated a synthetic PTM dataset that provides us with the ground truth cluster assignment. This dataset consists of five subsets, each having 100 peptides randomly picked from the yeast protein complex dataset, described in Sec. 4.3.3. Here, each set of peptides is assigned to have one of the five arbitrarily chosen modified amino acids: aspartic acid (D), phenylalanine (F), histidine (H), leucine (L), and proline (P). The true modification position for each peptide is randomly chosen to be on one of
Figure 4.5: Histograms of synthetic data modification position error. The histograms show the distribution of modification position error (the distance the true and predicted modification sites) for each of the five subsets of the synthetic PTM data with modified amino acids D, F, H, L and P, respectively. Combined with Fig. 4.6, the figures illustrate that identifying the set of PTM groups is nontrivial due to the noise present in the data.
the instances of the pre-assigned amino acid for that subset, and the modification positions used as input to the algorithms are set to a noisy version of the true modification positions. The noise (modification position errors) added are chosen from a standard normal distribution (see Fig. 4.5). Since the true modified amino acids are predefined, we can use them as labels to evaluate the performance of the algorithms.

The modification mass for each peptide is randomly generated to have Gaussian noise with a small variance (0.2) from the modification mass centre for the set of peptides that it belongs to (see Fig. 4.6). The distribution of modification masses was chosen to provide significant overlap in modification mass between adjacent sets. The modification mass centres were set to 40.0 Da for peptides with PTMs on D, 41.0 Da for peptides with PTMs on F, 42.0 Da for peptides with PTMs on H, 43.0 Da for peptides with PTMs on L, and 44.0 Da for peptides with PTMs on P.

For $k$-means clustering and MOG, the format of each input peptide is a vector consisting of the modification mass and the distance between the modification position and the closest instance of each amino acid, i.e., a vector of size 21 with the modification mass as the first element and the 20 amino acid as the next 20 elements (alphabetically ordered). The distance between the true modification position and each amino acid is used to account for our expectation that each PTM occurs on a specific set of amino acids. This means the true modified amino acid will more likely to have a small distance value.

### 4.3 Experimental Evaluations

We conducted two proof-of-concept experiments to validate the effectiveness of PTMClust. First, we compared PTMClust to two standard clustering algorithms: $k$-means clustering (Sec. 3.2.2) and MOG (Sec. 3.2.6), on a synthetically generated PTM dataset. Second, we benchmarked PTMClust against three state-of-the-art blind PTM search en-
Figure 4.6: Histograms of synthetic data modification mass. The histograms show the distribution of observed modification mass for each of the five subsets of the synthetic PTM data with modified amino acids D, F, H, L and P, respectively. This dataset is designed to have overlapping modification between different subset of the data. Combined with Fig. 4.6, the figures illustrate that identifying the set of PTM groups is nontrivial due to the noise present in the data.
gines and a PTM refinement algorithm on a reference phosphopeptide dataset. To show the strengths PTMClust, we applied it to process a yeast proteome dataset that contains multiple PTMs.

In our experiments, we initialized our algorithm with an initial number of clusters $K = 150$ (except for the first proof-of-concept experiment); the prior probability of each PTM group $\alpha_k = \frac{1}{K}$, where $k \in \{1, ..., K\}$; given that the PTM group is $k$, the probability that the PTM occurs on the $i$-th amino acid $\beta_{ki} = \frac{1}{A}$; the modification mass mean for each PTM group $\mu_k$ to be uniformly distributed across the searched modification mass range (except for the first proof-of-concept experiment); and the variance of modification mass for each PTM group $\Sigma_k = 1$ and limited, during learning, to be no greater than 2 to account for our assumption that the variance for each PTM group is small. The assumption on $\Sigma_k$ corresponds to our knowledge that for a PTM group to be physically relevant, it should have a well-defined modification mass.

### 4.3.1 Algorithm Comparison on Synthetic Data

Both $k$-means clustering and MOG are standard clustering methods because they perform effectively in many cases and are simple to understand and implement. Our algorithm improves upon $k$-means clustering and MOG by explicitly modelling the hidden relationships between the modification mass, modified amino acid, peptide sequence and modification position. We evaluated the performance of PTMClust against these two algorithms using a synthetic PTM dataset, which provides us with ground truth labels for the true modified amino acids, modification positions, modification masses, identities of the PTM groups and cluster assignment for each peptide. This synthetic data is designed to have overlapping modifications, in terms of modification masses and modified amino acids, so that it is nontrivial to identify the PTM groups. Since the goal of this experiment is to evaluate how well the three algorithms perform with increasing complexity in the input data, multiple datasets were generated with the number of PTM
groups ranging from two to five (see Sec. 4.2).

To test whether each method could identify the PTMs, we fixed the number of clusters ($K$) for each algorithm. In practice, PTMClust can automatically determine the number of PTM clusters, but we deactivated this feature for this experiment. The initial parameter setting for the modification mass cluster centres, shared for all three algorithms, was initialized randomly within the range of modification masses in the input dataset. For MOG, the variances were initialized to 1 for modification mass (consistent with PTMClust) and distances between the observed modification position and the closest instance of each amino acid (same variance used to generate the data). Theoretically, a large initial variance for modification mass (e.g., 10 in this experiment) can result in data points being falsely assigned to one cluster, because many data points with different labels have the same observed modified amino acids. This has an effect much like our background model. At the other extreme, a small initial variance for modification mass (e.g., 0.1) can cause clusters explaining only a few data points present near the initial cluster centres. However, due to the small size and simplicity of this dataset, we did not see significant differences in performance for both MOG and PTMClust as we varied the initial modification mass variances (data not shown). For each method, we performed 30 random restarts and picked the restart with the best joint log-likelihood. To do this, we learned $k$-means clustering by modifying the MOG algorithm (discussed in Sec. 3.2.6), where, after each EM iteration, we set the probability between a data point and its closest cluster centre to 1 and 0 for all other cluster centres to that data point.

Using $i \in \{1, ..., K\}$ to index each cluster, we evaluated the performance of the algorithms using a criterion that measures how well each ground truth PTM was detected. For cluster $i$, the largest group of peptides with the same label assigned to it was considered as true positives ($TP_i$) and all other peptides assigned to it were false positives ($FP_i$). To evaluate each algorithm, we calculated the correction rate ($CR$), which is defined as the difference between the total number of true positives and the total number
of false positives divided by the total number peptides in the sample summed over all clusters, given as
\[ CR = \frac{\sum_i TP_i - \sum_i FP_i}{N}, \]
where \( N \) is the total number of peptides in the sample. Intuitively, \( CR \) is a measure of the fraction of PTM predictions that are expected to be not due to chance.

Fig. 4.7 shows the result of applying the \( k \)-means clustering, MOG and PTMClust on input datasets with varying number of PTM groups. The figure shows that our algorithm outperforms both \( k \)-means clustering and MOG. Furthermore, PTMClust performs consistently well, while the performances of the other two algorithms exhibit a significant drop as the complexity of the dataset increases.

### 4.3.2 Benchmarking Against Phosphopeptide Predictions

We next examine the abilities of our algorithm to identify PTM groups corresponding to \textit{bona fide} PTMs, to fine-tune observed modification masses and to correct for misplaced modification positions. We chose to focus this analysis on phosphorylation since it plays a vital role in protein regulation for many different biological processes. As a result, it is well studied and annotated datasets are readily available. Using a dataset of ion trap MS/MS spectra (human HeLa cells) previously mapped and manually validated as phosphopeptides [189], we compared the initial PTMs identified from three state-of-the-art blind PTM search engines, SIMS [8], InsPecT [39, 78] and MODmap [190, 191] to the results after applying PTMClust on each of them individually. Additionally, we post-processing the result from InsPecT with the PTM refinement algorithm PTMFinder for the same dataset and compared the result against PTMClust. Fig. 4.8 shows a flowchart of how we generated the eight different outputs of ‘final’ PTM predictions taken from each PTM search engines and combinations of blind PTM search engines and PTM refinement.

\^\textsuperscript{3}The MS-alignment algorithm [78], which is part of the InsPecT program, was used to perform blind PTM search.
Chapter 4. Computational Refinement of PTM Predictions

Figure 4.7: Clustering algorithm comparison on a synthetically generated dataset. It shows how each of the three algorithms, \( k \)-means clustering, Mixture of Gaussians (MOG) and PTMClust (our algorithm), performs as more sets of data points with different modifications are added (increasing complexity). Correction rate (CR) is a quality measure defined as the difference between the total true positives and the total false positives divided by the total number peptides in the sample; higher correction rate indicates better performance. The result shows PTMClust performs consistently well while the other two algorithms exhibit a significant drop as the complexity of the dataset increases.
Figure 4.8: Phosphopeptide benchmark experiment flowchart. The flowchart shows how we generated the eight different outputs of ‘final’ PTM predictions taken from each PTM search engines and combinations of blind PTM search engines and PTM refinement algorithms used in this experiment. The general flow of the experiment starts with ion trap MS/MS spectra taken from [189] and ends with a set of PTM predictions from either a blind PTM search engine or one of two PTM refinement algorithms, PTMClust or PTMFinder. As part of the experiment, we obtained final PTM predictions from InsPecT, InsPecT post-processed with PTMClust, InsPecT post-processed with PTMFinder and InsPecT post-processed with PTMFinder and PTMClust. This last set of results allow us to directly compare the effectiveness of both PTM refinement algorithms, PTMClust and PTMFinder.

The dataset consists of 1655 spectra but we focused only on the 1340 spectra mapped and curated as singly modified phosphopeptides since SIMS, InsPecT and PTMClust are limited to one modification per peptide prediction. When searching the spectra, we used the default settings optimized for ion trap instruments for InsPecT, PTMFinder and MODmap, and reference settings described in [8] for SIMS. To simulate a true blind PTM search, an empty list of known PTMs was passed into PTMFinder, which ensures any corrections made by PTMFinder are not influenced by prior knowledge of known PTMs. Due to the long search time required, which scales linearly with the size of the
reference database, a common practice employed by blind PTM search engines [8, 39, 78] is use of a two-pass approach [192]. This two-pass approach involves 1) analyzing the spectra looking for only unmodified peptides and generating a reduced reference protein database containing proteins with at least one sequenced peptide, and 2) interpreting the spectra looking for modified peptides with this reduced reference protein database. The human database from the National Center for Biotechnology Information (NCBI) was used as the initial reference database in this two-pass approach. A reduced reference database of 1827 real proteins appended with the same number of decoy proteins and a common modification range [-20, 300] Da was used for all algorithms.

Amongst the 952 output peptide sequences matching the reference for SIMS, 267 had their modification misplaced. Similarly, InsPecT result matched 860 reference sequences but misplaced 239 modification positions. Using the default settings, MODmap produced a peptide sequence for only 157 spectra, which resulted in 125 peptide sequences matching to the reference with 28 of those having misplaced modification positions. Lastly, post-processing InsPecT result with PTMFinder produced a refinement to five peptide predictions: two peptide sequence changes resulted in a match to the reference but both cases failed to identify the correct modification position; an incorrect modification position change on a previously correct prediction; and two incorrect modification position changes on previously mismatch modification positions (i.e., no positive effect). In summary, we observed for PTMFinder 242 of the 862 peptide sequences matching to the reference with a misplaced modification position.

We initialized our algorithm as described above. Weighting the trade-off between maximizing RD and minimizing RFD, we settled on a model complexity setting of $\alpha^b = 0.90$, which resulted in a RD of 0.76 and a RFD of 0.27 for SIMS; $\alpha^b = 0.94$ with a RD of 0.72 and a RFD of 0.34 for InsPecT, $\alpha^b = 0.94$ with a RD of 0.72 and a RFD of 0.34 for PTMFinder; and, $\alpha^b = 0.45$ with a RD of 0.701 and a RFD of 0 for MODmap.

As shown in Table 4.1, PTMClust is able to correct a significant portion of the mis-
Table 4.1: Benchmark results between PTMClust, SIMS, InsPecT, MODmap and PTMFinder.

<table>
<thead>
<tr>
<th></th>
<th>No. of correct modification position matches (%)</th>
<th>No. of misplaced modification position matches (%)</th>
<th>Total correct peptide sequence matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMS</td>
<td>685</td>
<td>267</td>
<td>952</td>
</tr>
<tr>
<td>SIMS with PTMClust</td>
<td>791 (~15%)</td>
<td>161 (~40%)</td>
<td>952</td>
</tr>
<tr>
<td>InsPect</td>
<td>621</td>
<td>239</td>
<td>860</td>
</tr>
<tr>
<td>InsPect with PTMClust</td>
<td>712 (~15%)</td>
<td>148 (~38%)</td>
<td>860</td>
</tr>
<tr>
<td>PTMFinder</td>
<td>620</td>
<td>242</td>
<td>862</td>
</tr>
<tr>
<td>PTMFinder with PTMClust</td>
<td>711 (~15%)</td>
<td>151 (~38%)</td>
<td>862</td>
</tr>
<tr>
<td>MODmap</td>
<td>97</td>
<td>28</td>
<td>125</td>
</tr>
<tr>
<td>MODmap with PTMClust</td>
<td>108 (~11%)</td>
<td>17 (~39%)</td>
<td>125</td>
</tr>
</tbody>
</table>

A reference set of MS/MS spectra previously mapped to phosphopeptides [189] was analyzed by SIMS [8], InsPecT [39, 78], MODmap [191], and InsPecT follow by PTMFinder [86], a PTM refinement algorithm. Using the reference peptide sequences and modifications as the reference, the table shows the number of correct peptide sequence matches, and the correct and misplaced modification positions before and after applying PTMClust (our algorithm) onto the results from the four algorithms. PTMClust is able to correct for a significant portion of the modification position errors made by the four algorithms, and the improvements are consistent across different algorithms. Furthermore, PTMClust can make corrections to cases which PTMFinder missed, significantly outperforming it in terms of refining PTMs.
placed modifications identified by SIMS, InsPecT, MODmap and PTMFinder. Across the board, PTMClust performs consistently well with each tested algorithm. More specifically, for SIMS, PTMClust decreases the number of misplaced modifications by $\sim 40\%$ (106 fewer misplaced modification positions) to produce 791 correct matches, an increase of $\sim 15\%$. Similarity, for InsPecT, our algorithm reduces the number of misplaced modification positions by $\sim 38\%$ (91 fewer modification position misplacements) to produce 712 correct predictions, an increase of $\sim 15\%$. PTMClust obtains an improvement on par for MODmap with a $\sim 39\%$ decrease in the number of misplaced modifications (11 fewer misplaced modification positions) and a $\sim 11\%$ increase of correct predictions (108). For PTMFinder, we experience similar improvements to those for InsPecT, where we obtain $\sim 38\%$ (91) fewer modification position misplacements to produce 711 correct predictions, an increase of $\sim 15\%$.

A breakdown of the results shows that our algorithm made only few incorrect PTM refinements (19 for SIMS, 26 for InsPecT, 1 for MODmap and 26 for PTMFinder) where it incorrectly changed modification positions that were correctly identified by SIMS, InsPecT, MODmap or PTMFinder, while making a large number of improvements (125 for SIMS, 117 for InsPecT, 12 for MODmap and 117 for PTMFinder). A closer examination of the models learned for all four algorithms shows that the majority of the reference phosphopeptides are assigned to a PTM group with modification mass $\sim 79.87$ Da and high likelihood for S ($\sim 0.94$) and T ($\sim 0.06$). This observation corresponds correctly to our knowledge about phosphorylation that modifications occur on S and T and have a modification mass $\sim 80$ Da. A listing of the search results from all algorithms is provided in the supplementary table S1 in [12].

Next, we examined the overlap between the results from SIMS, InsPecT and MODmap, as well as the corrected results after applying our algorithm. PTMFinder is omitted here since its results are nearly identical to InsPecT. It has been previously reported that a significant portion of the results from SIMS and InsPecT do not match [8] (this is widely
believed to be true for many pairs of blind PTM search methods). Our analysis shows that many of the mismatches are due to incorrect modification position assignments: 229 of the 790 spectra that both SIMS and InsPecT mapped to the same peptide sequence have different assigned modification positions. After post-processing with our algorithm, \(\sim 41\%\) (93) of the mismatches was corrected, which significantly improved the overlap between the results from the two algorithms. We are able to observe similar improvements when we included MODmap into the analysis: 25 of 106 spectra have mismatching modification position with \(\sim 44\%\) (11) improvement between InsPecT and MODmap, while 25 of 98 spectra have mismatching modification position with \(\sim 48\%\) (12) improvement between all three algorithms (SIMS, InsPecT and MODmap). Due to the small number of observed mismatched modification positions amongst the overlaps between SIMS and MODmap (14 of 119 (\(\sim 12\%\)) matching peptide sequences), we did not observe any improvement post-processed with PTMClust. PTMClust consistently is able to improve on the overlap of the identified modified peptides between the different algorithms. This is not true in the case of SIMS vs MODmap because of there are too few overlaps. Our findings on this algorithm overlap analysis provide additional conference that PTMClust is producing sensible results.

4.3.3 Large-scale PTM Analysis of Yeast Proteome

To test its versatility in detecting diverse PTM groups in a more complex biological context, we applied PTMClust to analyze a large-scale PTM dataset taken from analyses of yeast protein complexes (LC-MS/MS spectra only) [121] using SIMS. The yeast dataset consists of over 2 million ion trap MS/MS spectra of which 19,560 putatively modified peptides (estimated false discovery rate of 4.3% based on the number of decoy peptides identified) were identified by SIMS with modification range (0, 200] Da. In this experiment, we used a model complexity setting of \(\alpha^b = 0.92\), which resulted in a RD of 0.58 and a RFD of 0.16.
Analysis with our algorithm is able to identify 121 PTM groups. The complete lists of modified peptide predictions are provided in the supplementary table S2 in [12] and a summary of the frequent PTMs observed are listed in the supplementary table S3 in [12]. Within this list of PTM groups are naturally occurring PTMs such as phosphorylation, acetylation and oxidation, and in vitro artificial modifications such as oxidized methionine and sodium/potassium salt adduct. Within the complete list of modified peptide predictions are many modified peptides not previously annotated to contain these modifications. In addition to those listed, there are many putative novel modification types that are worthy for further investigations.

To validate that our approach is generally applicable to any PTM, we compared the results before and after applying PTMClust to known modified yeast proteins taken from the Uniprot knowledgebase (Release 2010_11). A breakdown of our findings is shown in Table 4.2. For this analysis, we determined the modification sites (positions in the corresponding protein where the modifications occur) for each modified peptide in our results and matched them against the list of known modification sites from Uniprot. We found 213 modified peptide matches consisting of 75 unique known modification sites before, and 232 modified peptide matches and 81 unique modification sites after applying PTMClust, for an overall improvement of ∼9%. This analysis shows that PTMClust is able to detect and refine different kinds of PTMs, such as acetylation, cysteine oxidation (cysteine sulfenic acid) and phosphorylation.

A novel feature of PTMClust is the ability to consider modifications at the ends of proteins and peptides. Examples are modified peptides exhibiting N-terminus glycosylation (modification mass ∼162 Da) [193]. This modification is a PTM that adds sugar molecules to proteins and is known to play a vital role in proteolytic resistance, protein solubility, stability, local structure, lifetime in circulation and immunogenicity [194]. Although the original distribution of modified amino acids does not show any pattern with modifications mainly found on alanine (A), isoleucine (I), leucine (L) and valine (V),
Table 4.2: Summary of known modifications in the yeast proteome dataset.

<table>
<thead>
<tr>
<th>PTM</th>
<th>PTMClust</th>
<th>SIMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Known PTM Sites (% improvement over SIMS)</td>
<td>Peptides with Known PTM Sites (% improvement over SIMS)</td>
</tr>
<tr>
<td>Phosphorylation</td>
<td>66 (~8%)</td>
<td>115 (~15%)</td>
</tr>
<tr>
<td>Acetylation</td>
<td>9 (~13%)</td>
<td>75 (~42%)</td>
</tr>
<tr>
<td>Cysteine Oxidation</td>
<td>1 (~0%)</td>
<td>7 (~17%)</td>
</tr>
<tr>
<td>(Cysteine sulfinic acid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>5 (~0%)</td>
<td>35 (~0%)</td>
</tr>
<tr>
<td>Total</td>
<td>81 (~8%)</td>
<td>232 (~9%)</td>
</tr>
</tbody>
</table>

The known set of modifications was taken from Uniprot (Release 2010_11). We matched the sets of modified peptides produced by SIMS and post-processed with PTMClust to the set of known yeast modification sites. The results show PTMClust is able to identify and refine PTMs in a complex dataset.
PTMClust is able to recognize that all the modifications occur close to the N-terminus of the peptide. This observation is unlikely to be explained by simple amino acid substitutions or artifacts since the modified peptides have a similar initial modification mass, and their modification was initially observed to occur on different amino acids. In terms of where the modifications occur, they all share the commonality that their modifications occur at near the N-terminus, which PTMClust can capture. Therefore, this observation is highly unlikely to occur by chance. N-terminus glycosylation is an example that illustrates PTMClust is able to find terminus PTMs.

4.4 Summary and Future Directions

Accurate identification of protein modifications in protein sequences is a critical first step in any PTM study, and thus it may benefit the utility of proteomic profiling to address research problems in basic biology, as well as biomarker discovery and drug development in the clinical domain. A recently developed approach for PTM discovery is to analyze MS/MS data using a blind PTM search method. Genome-wide studies using SIMS, InsPecT and other blind PTM search engines have reported numerous PTM candidates [8, 39, 78, 80, 81, 82]. However, these search methods suffer from two problems: mass measurement inaccuracy and uncertainty predicting modification positions, which limit their accuracy and precision. We developed a novel machine learning algorithm called PTMClust for post-processing the results of blind PTM search engines to improve prediction confidence. Our method works by simultaneously identifying the positions of the most likely modified amino acids and grouping peptides with similar modification mass and modified amino acid side chains. We demonstrated that PTMClust improved on both true positives (correct modification position predictions) and false positives (misplaced modification positions) when applied to the output from SIMS, InsPecT, MODmap, and InsPecT post-processed with PTMFinder, a PTM refinement algorithm.
The results show that our algorithm was able to detect a number of previously-annotated naturally occurring and artificially induced PTMs, most notably phosphorylation, but also acetylation (lysine), oxidization (methionine), and the formation of non-covalent adducts (e.g., sodium/potassium salts). In addition, our algorithm facilitates the identification of terminal modifications, which is a feature not currently found in common blind PTM search engines. To our knowledge, PTMClust is the first technique to systematically and objectively address sequence-dependent variation in the PTM dataset at the PTM level, and improve the reliability of individual PTM identification.

For the task of PTM refinement, we have shown that PTMClust outperforms PTMFinder on the dataset of phosphopeptides. The reason for the poor performance for PTMFinder is because there are only $69 (\sim 4\%)$ of the total peptides that are duplicates, i.e., there is another instance with the same peptide sequence and similar modification mass. This is expected since it is known only a small portion of spectra in an experiment are mapped to modified peptides and current MS experimental protocols for genome-wide studies are designed to sample as many different peptides as possible (through the use of an exclusion list in the mass spectrometer). Moreover, many instances of the same modified peptide either share the same modification position (for both correct and misplaced) or have a vastly different modification position that points to another phosphorylation site in the peptide. The former can be explained since same missing peaks due to incomplete fragmentation are, in general, consistently not detected across different instances of the same peptide sequence, and blind PTM search algorithms produce the same modified peptide prediction for similar-look spectra. For blind PTM searches, PTMFinder only works when there are multiple instances of the same modified peptide. On the other hand, PTMClust is successful even for low abundance modified peptides as long as there are multiple instances of the same underlying PTM.

We believe PTMClust is complementary to, and can benefit from technological improvements in mass spectrometer instrumentation. Two of the more prominent advance-
ments in recent years are high mass accuracy and alternate fragmentation mechanisms. For high mass accuracy mass spectrometers, such as an Orbitrap [195], mass errors are significantly reduced, and peak intensity signal-to-noise ratios are greatly improved in the observed MS/MS spectra if they are acquired in high resolution mode. However, currently the common practice for experiments using Orbitrap is to generate MS/MS spectra in low resolution mode due to its higher scan rate. Distinguishing features of electron-transfer dissociation (ETD), a recently introduced fragmentation mechanism, are its abilities to preserve the localization of labile PTMs and produce near complete ion fragmentation [85]. The problem is ETD only works on peptides with charge state greater than +2 and can identify significantly fewer peptides than other fragmentation methods. To address these issues, current approach is to use a mass spectrometer equipped with ETD and another fragmentation method, such as CID, and switch between them depending on the properties of the peptides to be fragmented [196, 197]. These technological advancements can help reduce the issue of misplaced modification position due to missing peaks and noisy spectra but can still benefit from using PTMClust in its analysis. Given input data with higher mass resolution and fewer misplaced modifications due to cleaner ion fragmentation signal, PTMClust can improve upon its abilities to refine modification positions and find meaningful PTM groups. Using our algorithm, we plan to analyze modified peptides processed from spectra generated by both low and high resolution mass spectrometers using a variety of fragmentation methods (e.g., CID, ETD and high-energy collision dissociation (HCD)).

PTMClust has a number of weaknesses, similar to other statistical methods, which can potentially be solved by, for example, updating the model. Although RD and RFD can provide a confident estimate for the overall result, our algorithm does not provide a confidence score per peptide and per modification, which is a feature that can be found in PTMFinder. Additionally, it cannot detect PTMs that occur only once in the data, since multiple instances are needed for model building. Moreover, it is unable to handle more
than one modifications per input sequence. Furthermore, the model parameter maximum number of PTM groups needs to be expertly define since if set inappropriately then the algorithm can produce erroneous results. Lastly, depending on the mass resolution in the input data, PTM groups identified by our algorithm may contain multiple PTMs with similar modification mass. Despite these limitations, we are able to obtain results exceeding the performance of the state of the art, and show the utility, reliability and generality of our algorithm in refining PTMs found by blind PTM search engines.

Lastly, there are two noteworthy extensions that we like to highlight. First, we suggest combining blind search algorithms with our algorithm to jointly analyze MS data for modified and unmodified peptides. This allows our algorithm to take into account ion fragmentation patterns directly. One advantage to this extension is it can handle cases where multiple equally likely modification positions are present in the peptide, but the modification was originally misplaced. An example would be multiple serines appearing side-by-side in the peptide and the modification (phosphorylation) was original misplaced on one of the serine. Second, we recommend exploring the use of nonparametric Bayesian approaches to allow for automatic model selection, which we have implemented and is presented in Chap. 5. In this improved PTM refinement algorithm, we modelled our proposed PTM generative process with a variant of an IMM (Sec. 3.2.10) that can capture the data more precisely. This approach has additional advantages of improved PTM refinement capabilities and does not need any manual parameterization.
Chapter 5

Nonparametric Bayesian Approach to PTM Refinement

Although current state-of-the-art blind PTM spectral analysis algorithms can predict thousands of modified peptides (PTM predictions) in an MS/MS experiment, a significant percentage of these predictions has inaccurate modification mass estimates and false modification site assignments [12]. This problem can be addressed by post-processing the PTM predictions with a PTM refinement algorithm. However, PTM refinement has been largely overlooked in the field of MS/MS analysis. In Ch. 4, we presented a machine learning algorithm, PTMClust, for refining PTM predictions obtained from blind PTM search engines to improve PTM prediction quality. Regarding the number of correct PTMs found, we show that PTMClust outperforms both the blind PTM search engines alone and post-processed with state-of-the-art PTM refinement algorithm PTMFinder. Even so, PTMClust has a small number of limitations and areas for improvement.

In this chapter, we address several limitations of PTMClust: lack of confidence score per modification site, manual parameterization and the use of a greedy-based, non-automatic model selection algorithm. We overcome these issues by extending the PTMClust model to allow for an unbounded number of mixture components that can ac-
count for uncertainties in the quantity and identity of PTMs in the input data. Our final model is similar to IMMs described in Sec. 3.2.10 where each of the unlimited number of mixture components is modelled similar to a mixture component in PTMClust. Therefore, our new model can be viewed as an infinite PTMClust mixture model (iPTMClust). Much like other IMMs, iPTMClust uses MCMC sampling methods (see Sec. 3.2.11) for inference. The benefits of iPTMClust are that it 1) achieves significant improvement over other PTM refinement algorithms, including PTMClust, 2) provides a confidence score per modification position and 3) removes the need for any manual parameter tuning. The latter point is important because it makes iPTMClust practical for non-experts and, hence, simple to use.

The primary goal behind iPTMClust is to improve the quality of the PTM predictions. First, to demonstrate that iPTMClust produces sensible and accurate cluster assignments, we compare it to $k$-means clustering (Sec. 3.2.2), MOG (Sec. 3.2.6) and PTMClust on a synthetically generated PTM dataset. Second, in two separate benchmark experiments using PTM data taken from a phosphopeptide and a yeast proteome study, respectively, we show that iPTMClust outperforms state-of-the-art PTM prediction and refinement algorithms, including PTMClust. Finally, we illustrate the general applicability of our new approach on a set of human chromatin protein complex data, where we are able to identify putative novel modified peptides and modification sites that may be involved in the formation and regulation of protein complexes. It is known that PTMs, such as phosphorylation, play a vital role in regulating protein-protein interactions. Accurate PTM profiling is an important step in understanding the mechanisms behind many biological processes and should be an integral part of any proteomic study.

In this chapter, we begin by providing the motivation behind and the contribution of our new algorithm iPTMClust. Next, we describe our modelling approach, present a graphical representation of the model and its description, and derive the two MCMC sampling algorithms, the split-merge Metropolis-Hastings and the Gibbs sampling, used
to perform inference. Lastly, we present benchmark results, which show that our new algorithm outperform current state-of-the-art algorithms, and an application on a human protein complex study that highlights the importance of accurate PTM predictions. We conclude with a summary and suggestions for future research directions.

5.1 Motivation and Background

PTMs are known to play a vital role in the cell, and are proven to be instrumental in many disease-related studies as discussed previously in Ch. 2. A core task in studies involving PTMs is PTM prediction, i.e., identification of peptide sequences and PTMs associated with each modified peptide within a biological sample. A preferred experimental procedure for PTM prediction is MS/MS followed by an analysis with a blind PTM search engine (see Sec. 2.3). Blind PTM search engines are commonly used because of their versatility to account for both known and novel PTMs. However, we have argued in Ch. 2 that PTM prediction using blind PTM search engines alone is insufficient since a significant percentage of predicted PTMs from blind PTM search engines contain inaccurate modification masses and incorrect modification positions. It is prudent to incorporate PTM refinement as part of a PTM prediction pipeline, as it can significantly improve the quality of PTM predictions. We also demonstrated that post-processing with PTMClust (our previous PTM refinement algorithm) greatly improves the number of positive predictions while reducing the amount of false PTM assignments. Furthermore, the result of our benchmark experiments indicate that PTMClust significantly outperforms PTM search engines alone and the state-of-the-art PTM refinement algorithm, PTMFinder.

The principle behind and the distinguishing feature of PTMClust is modelling modifications at the PTM level instead of at the peptide level. This approach has the advantage of allowing the model to account for low abundance modified peptides since other peptides with the same underlying PTM can help identify the correct but unknown modification
mass and modified amino acid. PTMClust uses a generative model to capture the hidden relationship between factors influencing the PTM mapping process. PTMClust uses the EM algorithm and a modified version of the split and merge model selection method to learn and infer an optimal parameter setting for the model. As part of the model selection procedure, a range of models are learned by adjusting a model complexity parameter, and the final model is selected manually by weighting the trade-off between false positives (decoy peptides) allowed and real peptides detected. Despite this cumbersome procedure, the resulting PTM predictions are of higher quality than those taken from existing blind PTM search engines alone or post-analyzed with PTMFinder.

Although it produces class-leading results, PTMClust has its limitations. In our new algorithm iPTMClust, we set out to address three specific drawbacks of PTMClust: 1) the use of a greedy-based, non-automatic model selection algorithm, 2) the need for manual parameter tuning and 3) the lack of a confidence score per modification position. We overcame these issues by extending the PTMClust model to allow for an unbounded number of mixture components that can account for uncertainties in the quantity and identity of PTMs in the input data. This extension parallels the conversion from a finite to an infinite mixture model, which we outlined in Sec. 3.2.10 but the complex nature of the underlying PTMClust model makes this extension nontrivial. We derived and implemented the split-merge Metropolis-Hastings (Sec. 3.2.11) and the Gibbs sampling algorithm (Sec. 3.2.11) for our model to efficiently infer the groupings of input modified peptides and refine the peptides’ modification masses and modification positions. At the end, iPTMClust achieves the following benefits: 1) outperforming PTMClust and other PTM refinement algorithms, 2) providing a fully-automated model selection method without the need for any manual parameterization and 3) offering modification position level confidence scores that users can use to assess the quality of the result and further refine their analyses.
5.2 Methods

Similar to our first algorithm, at the heart of iPTMClust is a generative probability model that describes a process in which observed modified peptides can be generated by modelling the complex interactions between hidden variables that play a role in the protein modification process. Given an observed modification, we assume it comes from one of many PTMs. However, the number and identity of these PTMs are unknown. Our method accounts for this uncertainty by considering as many different PTMs as needed, represented by an infinite number of PTM groups. By defining appropriate priors on the hidden variables, over-fitting can be avoided, and only a finite set of PTM groups are used at anytime during inference. The latter point is important because it makes calculations in the algorithm tractable. During inference, the properties of the active PTM groups are influenced by the input data and the chosen priors. We adapted both the Gibbs sampling and the restricted Gibbs sampling split-merge algorithms to infer the values of the hidden variables and parameters in our model. After inference, these hidden variables and model parameters can be used to deduce the true modification mass and a confidence score per possible modification position for each input peptide sequence, or the MAP estimate of modification masses and positions.

5.2.1 iPTMClust Algorithm

The core of the generative model in iPTMClust is the same as in PTMClust: it describes how a pair of observed modification mass and modification position are generated. The difference between iPTMClust and PTMClust is the inclusion of priors on model variables and parameters that govern the choice of active PTM groups from a boundless number of PTM groups. Given the type of PTM (PTM group) chosen from one of the limitless numbers of PTM groups, we can generate the observed modification mass as a noisy version of the modification mass mean, and select an amino acid most likely to contain
the modification as the modified amino acid. Given the peptide sequence, we can choose as the ‘true’ modification position a position uniformly along it that matches the modified amino acid. Finally, we assume that the observed modification position is a noisy version of the ‘true’ modification position.

The structural relationships between variables are shown by the Bayesian network in Fig. 5.1. The top part outlines the priors represented by their corresponding hyperparameters placed on the model parameters: mixing coefficient, modification mass means, modification mass variances and probability of modification occurring on an amino acid. The bottom portion describes the model for one input peptide and is repeated for \( N \) inputs, as indicated by the plate notation; this portion is the same as PTMClust.

The graphical representation of our model resembles that of a simplified IMM shown in Fig. 3.10(c). Here, we explicitly display each model parameter and its hyper-parameters instead of abstracting them as parameters \( \theta \) and priors \( G_0 \), respectively. In the following, we outline the priors and hyperpriors placed on model parameters that are new in this model in more detail. The rationale and intuition behind the bottom section is given in Sec. 4.2.

In our model, each input peptide sequence \( S_n \), indexed by \( n \in \{1, \ldots, N\} \), where \( N \) is the number of peptides in the dataset, has a corresponding discrete peptide length \( L_n \), observed modification position \( x_n \in \{1, \ldots, L_n\} \), and observed modification mass \( m_n \). We denote the amino acid in position \( j \) of the input sequence \( n \) as \( S_n(j) \). The total number of values \( S_n(j) \) can take on is \( A = 24 \), which includes the 20 naturally-occurring amino acids and four special characters indicating the beginning and end of proteins and peptides. Additionally, we denote \( o_k \) to be the number of input peptides assigned to cluster \( k \). The hidden variable \( c_n \in [1, \ldots, \infty] \) denotes the unknown PTM group that peptide sequence \( n \) is assigned to. Given a CRP prior for the model, its probability conditioned on all other
Figure 5.1: A Bayesian network describing the generative model for our new algorithm $i$PTMClust, using plate notation. The shaded nodes represent observed variables, the unshaded nodes represent hidden variables and the variables outside the plate are model parameters. The model describes how the observed modification mass and the modification position are generated. The bottom part resembles PTMClust as it captures the assumption on how each observation is generated. The plate notation indicates that there are $N$ copies of the model, one for each input data. The top portion outlines the structure of the hierarchy of priors and hyperpriors placed on the model parameters, mixing coefficients, modification mass means, modification variances and probability of modified amino acid. The outer plate shows that there are infinite copies, one for each possible PTM group. The probability of modified amino acid $\beta_{kj}$ is embedded in two plates signifies that there are $K \times A$ copies, one for each $K \to \infty$ PTM groups and $A$ possible amino acids. The variables outside the plates are hyperpriors’ parameters.
Chapter 5. Nonparametric Bayesian Approach to PTM Refinement

The probabilistic model of the conditional posterior distribution of \(c_n\)’s is given as

\[
P(c_n = k|c_{N\backslash n}, \gamma) = \begin{cases} 
\frac{o_{-n,k}}{N-1+\gamma} & \text{if } o_{-n,k} > 0, \\
\frac{\gamma}{N-1+\gamma} & \text{otherwise},
\end{cases}
\]  

(5.1)

where \(o_{-n,k}\) is the number of peptides assigned to cluster \(k\) that does not consider \(n\)-th peptide sequence, \(N\backslash n\) indicates all indices excluding \(n\) and \(C_{N\backslash n}\) is shorthand notation for \(C_i : \forall i \in \{N\backslash n\}\). Eq. 5.1 follows from the derivation of IMMs with a CPR prior in Sec. 3.2.10. Furthermore, we can write the general form of Eq. 5.1 that takes into consideration the likelihood of the \(n\)-th input modified peptide belonging to the PTM group index by \(c_n\) as follows:

\[
P(c_n = k|c_{N\backslash n}, \gamma, \Theta) = \begin{cases} 
b\frac{o_{-n,k}}{N-1+\gamma} P(a_n, z_n, x_n, m_n|c_n, S_n, \Theta) & \text{if } o_{-n,k} > 0, \\
b\frac{\gamma}{N-1+\gamma} \int P(a_n, z_n, x_n, m_n, \mu, \Sigma, \beta|c_n, S_n, \Theta) \partial H_0(\mu, \Sigma, \beta) & \text{otherwise},
\end{cases}
\]  

(5.2)

where \(H_0\) indicates the prior distribution placed on \(\mu, \Sigma\) and \(\beta\), \(b\) is the appropriate normalizing constant so the probabilities sum to one, \(\gamma\) is the hyper-parameter concentration parameter and \(\Theta\) represents hyper-parameters \(\lambda, \upsilon, \varphi, \xi, \gamma\) and \(\omega\). The calculations of the conditional posterior probability \(P(a_n, z_n, x_n, m_n|c_n, S_n, \Theta)\) and the integral \(\int P(a_n, z_n, x_n, m_n, \mu, \Sigma, \beta|c_n, S_n, \Theta) \partial H_0(\mu, \Sigma, \beta)\) are given later in Eq. 5.29 and Eq. 5.30 respectively. The hyper-parameter \(\gamma\) is given a vague inverse gamma prior, shown as

\[
P(\gamma) = IG(1, 1),
\]

\[
\propto \gamma^{-\frac{3}{2}} \exp \left( \frac{-1}{2\gamma} \right).
\]  

(5.3)

The conditional posterior of \(\gamma\) can be derived from expanding Eq. 5.1 to a joint distribution of \(c_n\)’s, and then combining it with the prior from Eq. 5.3 to give

\[
P(\gamma|k, N) \propto \gamma^{k-\frac{3}{2}} \exp \left( \frac{-1}{2\gamma} \right) \frac{\Gamma(\gamma)}{\Gamma(N + \gamma)}.
\]  

(5.4)
We assume that the observed modification mass for each PTM group is noisy and modelled it to be normally distributed around the true modification mass, given as

\[
P(m_n|c_n = k) = \mathcal{N}(\mu_k, \Sigma_k) = \frac{1}{\sqrt{2\pi \Sigma_k}} \exp \left( -\frac{(m_n - \mu_k)^2}{2\Sigma_k} \right),
\]

where \(\mu_k\) and \(\Sigma_k\) are the parameters modification mass means and variances for the \(k\)-th PTM group. The means, \(\mu_k\), are given Gaussian priors, written as

\[
P(\mu_k|\lambda, \upsilon) = \mathcal{N}(\lambda, \upsilon).
\]

The mean \(\lambda\) and variance \(\upsilon\) of these Gaussians are hyper-parameters common to all PTM groups. The hyper-parameters themselves are given vague normal and gamma priors as follows:

\[
P(\lambda) = \mathcal{N}(\mu_*, \sigma_*^2),
\]

\[
P(\upsilon) = \mathcal{G}(1, \sigma_*^2)
\]

\[\propto \upsilon^{-1/2} \exp(-\upsilon \sigma_*^2/2),\]

where \(\mu_*\) and \(\sigma_*^2\) are the mean and variance of the observed modification masses. The shape parameter of the gamma prior is set to 1, which corresponds to a very broad distribution. The variances, \(\Sigma_k\), whose inverse are given gamma priors, can be written as

\[
P(\Sigma_k^{-1}|\varphi, \xi) = \mathcal{G}(\varphi, \xi^{-1}),
\]

whose shape \(\varphi\) and scale \(\xi^{-1}\) are hyper-parameters common to all components. The hyper-parameters \(\varphi\) and \(\xi\) are given priors of inverse gamma and gamma on them, retrospectively, as follows:

\[
P(\varphi) = \mathcal{IG}(\mu_*, \sigma_*^2)
\]

\[\propto \varphi^{-3/2} \exp \left( \frac{1}{2\varphi} \right),\]

\[
P(\xi) = \mathcal{G}(1, \sigma_*^2).
\]
The conditional posterior distribution for $\mu_k$ is obtained by multiplying the likelihood of the observed modification mass from Eq. 5.5 conditioned on the cluster assignment, by the prior in Eq. 5.6 to give

$$P(\mu_k|c_n,m_n,\Sigma_k,\lambda,\upsilon) = N\left(\frac{\bar{x}_m o_{-n,k} \Sigma_k^{-1} + \lambda \upsilon}{o_{-n,k} \Sigma_k^{-1} + \upsilon}, \frac{1}{o_{-n,k} \Sigma_k^{-1} + \upsilon}\right), \quad (5.12)$$

where $\bar{x}_m = \left(\frac{1}{o_{-n,k}}\right) \sum_{i:c_i=m} x_i$ is the mean of the observed modification masses and $o_{-n,k}$ is the number of peptides excluding the $n$-th peptide belonging to cluster $k$. For the hyper-parameters, Eq. 5.6 plays the role of the likelihood, which together with priors given in Eq. 5.7 and Eq. 5.8 gives conditional posteriors of the form

$$P(\lambda|\mu_1,\ldots,\mu_k,\upsilon) = N\left(\frac{\mu^*/\sigma^2 + \upsilon \sum_{j=1}^k \mu_j}{1/\sigma^2 + k\upsilon}, \frac{1}{1/\sigma^2 + k\upsilon}\right), \quad (5.13)$$

$$P(\upsilon|\mu_1,\ldots,\mu_k,\lambda) = \mathcal{G}\left(k + 1, \left[\frac{1}{k+1} \left(\sigma^2 + \sum_{j=1}^k (\mu_j - \lambda)^2\right)\right]\right). \quad (5.14)$$

Similarly, the conditional posterior distributions for the variances are obtained by multiplying the likelihood of the observed modification mass from Eq. 5.5 conditioned on the cluster assignment, by the prior in Eq. 5.9 to give

$$P(\Sigma_k|c_n,m_n,\mu_k,\varphi,\xi) = \mathcal{G}\left(\varphi + o_{-n,k}, \left[\frac{1}{\varphi + o_{-n,k}} \left(\xi \varphi + \sum_{i:c_i=k} (m_n - m\mu_k)^2\right)\right]\right). \quad (5.15)$$

For the related hyper-parameters, Eq. 5.9 plays the role of the likelihood and together with priors given in Eq. 5.10 and Eq. 5.11 gives

$$P(\varphi|\Sigma_1,\ldots,\Sigma_k) \propto \Gamma\left(\frac{\varphi}{2}\right)^{-k} \exp\left(\frac{1}{2\varphi}\right) \left(\frac{\varphi}{2}\right)^{k-3/2} \prod_{j=1}^k \left(\Sigma_j\xi\right)^{-\frac{k-1}{2}} \exp\left(-\frac{\varphi \Sigma_j \xi}{2}\right), \quad (5.16)$$

$$p(\xi|\Sigma_1,\ldots,\Sigma_k,\varphi) = \mathcal{G}\left(k\varphi + 1, \left[\frac{1}{k\varphi + 1} \left(\sum_{j=1}^k \Sigma_j\right)^{-1}\right]\right), \quad (5.17)$$

where $\Gamma(\cdot)$ is the gamma function with the form

$$\Gamma(n) = (n-1)!$$
for a positive integer $n$.

By using conjugate priors on $\mu_k$ and $\Sigma_k$, both these parameters can be integrated out to give a probability of $m_n$ based directly on the hyper-parameters, given as

$$P(m_n|c_n, \lambda, \nu, \phi, \xi) = \int P(m_n|c_n = k, \mu_k, \Sigma_k)P(\mu_k|\lambda, \nu)P(\Sigma_k|\phi, \xi) \partial \mu_k, \Sigma_k$$

$$= t\left(\hat{\phi}, \hat{\lambda}, \frac{\hat{\nu} + 1}{\hat{\nu} + o_k} \hat{\xi}\right), \quad (5.18)$$

where $t(\cdot)$ is the student’s t-distribution, $o_k$ is the number of peptides assigned to cluster $k$, $\hat{\phi} = \frac{\phi \lambda + (o_k \bar{m}_k)}{\phi + o_k}$, $\hat{\lambda} = \lambda + o_k$, $\hat{\nu} = \nu + o_k$, $\hat{\xi} = \xi + \sum_{i:\forall i, c_i = k} (m_i - \bar{m}_k)^2 + \frac{\lambda o_k (m_k - \phi)^2}{\lambda + o_k}$, and $\bar{m}_k = \frac{1}{o_k} \sum_{i:\forall i, c_i = k} m_i$ is the average observed modification mass for peptides assigned to $k$-th PTM group.

Let $a_n : n \in \{1, ..., N\}$ denote the true (hidden) modified amino acid (i.e., the amino acid that the PTM occurs on) for the $n$-th peptide sequence. Then, the probability of $a_n$’s given that the PTM group is $k$ is given as

$$P(a_1, ..., a_N|c_n = k, \beta_{k1}, ..., \beta_{kA}) = \prod_{i=1}^A (\beta_{ki})^{o_{ki}}, \quad (5.19)$$

where $o_{ki} = \sum_{n \in o_k} \delta_{\text{Kronecker}}(a_n, i)$ represents the number of peptides that are assigned to cluster $k$ and have true modified amino acid $i$, $\delta_{\text{Kronecker}}(a, b)$ is the Kronecker delta function which returns 1 if $a = b$ and 0 otherwise, and the $\beta_{ki}$’s satisfy the constraints $\beta_{ki} \geq 0$ and $\sum_{i=1}^A \beta_{ki} = 1$. A dirichlet distribution prior, which is the conjugate prior, with hyper-parameter $\omega$ is given to the $\beta_{ki}$’s, shown as

$$P(\beta_{k1}, ..., \beta_{kA}|\omega) = Dir(\omega/A), \quad (5.20)$$

$$= \frac{\Gamma(\omega_i)}{\Gamma\left(\frac{\omega}{A}\right)} \prod_{i=1}^A (\beta_{ki})^{\frac{\omega_i}{A} - 1}.$$ 

The hyper-parameter $\omega$ is given a vague inverse gamma distribution

$$P(\omega) = IG(1, 1), \quad (5.21)$$

$$\propto \omega^{-3/2} \exp(-1/(2\omega)).$$
Since a conjugate prior is used on $\beta_{ki}$’s, we integrate out the $\beta_{ki}$’s using a standard Dirichlet integral to write the probability of $a_n$’s given the hyperprior $\omega$ directly as follows:

$$P(a_1, ..., a_N| c_n = k, \omega) = \int P(a_1, ..., a_N, | c_n = k, \beta_{k1}, ..., \beta_{kA}) \, \partial \beta_{k1}, ..., \beta_{kA}$$

$$= \frac{\Gamma(\omega)}{\Gamma\left(\frac{\omega}{A}\right)} \int \prod_{i=1}^{A} \left(\beta_{ki}\right)^{o_{ki} + \frac{\omega}{A} - 1} \, \partial \beta_{ki} \tag{5.22}$$

$$= \frac{\Gamma(\omega)}{\Gamma(o_k + \omega)} \prod_{i=1}^{A} \frac{\Gamma(o_{ki} + \frac{\omega}{A})}{\Gamma\left(\frac{\omega}{A}\right)}.$$

The conditional prior for each $a_n$ given all the others can be obtained from Eq. 5.22 by keeping all but one $a_n$ fixed, given as

$$P(a_n = i| a_{-n}, c_n = k, \omega) = \frac{o_{-n,ki} + \frac{\omega}{A}}{o_k - 1 + \omega},$$

where $a_{-n}$ are all $a$’s excluding $a_n$ and $o_{-n,ki}$ is number of peptides excluding the $n$-th input peptide assigned to PTM group $k$ and have $a_n = i$. This is exactly the same as the conditional probability of a cluster assignment given all others in an IMM using the CPR prior, given in Eq. 3.45 on p.85. Similarly, the conditional posterior for each $a_n$ can be written as

$$P(a_n = i| a_{-n}, c_n = k, z_n, x_n, S_n, \omega) = \frac{o_{-n,ki} + \frac{\omega}{A}}{o_k - 1 + \omega} P(z_n, x_n| a_n = i, S_n), \tag{5.23}$$

where $P(z_n, x_n| a_n = i, S_n)$ can be factorized into $P(z_n| a_n = i, S_n)P(x_n| s_n)$. Both $P(z_n| a_n = i, S_n)$ and $P(x_n| s_n)$ are discussed below. The conditional posterior for the hyper-parameter $\omega$ can be derived from Eq. 5.22 together with the prior from Eq. 5.21 to give

$$P(\omega| A, N) = \frac{\omega^{A-3/2} \exp\left(\frac{1}{2\omega} \Gamma(\omega)\right)}{\Gamma(N + \omega)}. \tag{5.24}$$

Lastly, by using the conjugate prior on $\beta_{ki}$’s (see Eq. 5.20), $\beta_{ki}$’s can be integrated out to give a probability of each $a_n$ based directly on the hyper-parameters, given as

$$P(a_n = i| c_n = k) = \int P(a_n| c_n = k, \beta_{ki}) P(\beta_{ki}| \omega) \, \partial \beta_{ki} \tag{5.25}$$

$$= \text{Dir}(\omega + o_{ki}).$$
The details of the rest of the model are the same as those in PTMClust given in Sec. 4.2. To reiterate, given the peptide sequence \( S_n \) and \( a_n \), we modelled each occurrence of \( a_n \) in the peptide sequence to have the same probability of being the true (unobserved) modification position \( z_n \), which can be written as

\[
P(z_n = j | a_n = i, S_n) = \begin{cases} 
\frac{1}{\delta_{ni} + 1} & \text{if } S_n(j) = i, j > 0, \\
\frac{1}{\delta_{ni} + 1} & \text{if } j = 0, \\
0 & \text{otherwise,}
\end{cases} \tag{5.26}
\]

where \( \delta_{ni} \) denotes the number of times amino acid \( i \) occurs in sequence \( n \) and \( z_n = 0 \) indicates that the true PTM occurs outside of the given peptide sequence. Given the true modification position \( z_n \), the probability of modification position error \( (x_n - z_n) \) for the observed modification position \( x_n \) is modelled with a discrete probability distribution, given as

\[
P(x_n | z_n = j) = \begin{cases} 
\phi(x_n - j) & \text{if } j > 0, \\
\phi(L_n) & \text{if } j = 0, \\
0 & \text{otherwise,}
\end{cases} \tag{5.27}
\]

where the likelihood function \( \phi \) accounts for the modification position error. This likelihood function is shared across all PTM groups and was inferred from our empirical observation of the yeast PTM dataset as described in Sec. 4.2. The conditional joint probability \( P(z_n, x_n | a_n, S_n) \) is simply the product of Eq. 5.26 and Eq. 5.27, given as

\[
P(z_n = j, x_n | a_n = i, S_n) = P(z_n = j | a_n = i, S_n)P(x_n | z_n = j),
\]

\[
= \begin{cases} 
\frac{\phi(x_n - j)}{\delta_{ni} + 1} & \text{if } S_n(j) = i, j > 0, \\
\frac{\phi(L_n)}{\delta_{ni} + 1} & \text{if } j = 0, \\
0 & \text{otherwise.}
\end{cases} \tag{5.28}
\]

Finally, we outline how the conditional posterior probability \( P(a, z, x, m | c, S, \Theta) \) and the integral \( \int P(a, z, x, m, \mu, \Sigma, \beta | c, S, \Theta) \partial H_0(\mu, \Sigma, \beta) \) used in Eq. 5.2 can be evaluated

\(^1\) \( z_n = 0 \) is needed to avoid numerical issues since our algorithm considers each amino acid as a possible modification target.
analytically. Based on the structure of the Bayesian network of our model given in Fig. 5.1 on p. 133 for a given PTM group \( k \), \( P(a, z, x, m|S, \Theta) \) can be derived as follows:

\[
P(a_n, z_n, x_n, m_n|c_n, S_n, \Theta) \\
= \int P(a_n, z_n, x_n, m_n, \mu, \Sigma, \beta|c_n, S_n, \Theta) \, \partial H_{-n,k}(\mu, \Sigma, \beta) \\
= P(z_n|a_n, S_n)P(x_n|z_n) \int P(a_n|c_n, \beta)P(\beta|\omega) \, \partial \beta \\
\int P(m_n|c_n, \mu, \Sigma)P(\mu|\lambda, \upsilon)P(\Sigma|\phi, \xi) \, \partial \mu, \Sigma,
\]

where \( H_{-n,k} \) is the posterior distribution of \( \mu, \Sigma \) and \( \beta \) based on their priors and all peptides that are assigned to the \( k \)-th PTM group excluding the \( n \)-th peptide sequence. We can then substitute the evaluations of the integrals from Eq. 5.18 and Eq. 5.25 to give

\[
P(a_n, z_n, x_n, m_n|c_n, S_n, \Theta) = \int P(a_n, z_n, x_n, m_n, \mu, \Sigma, \beta|c_n, S_n, \Theta) \, \partial H_{-n,k}(\mu, \Sigma, \beta) \\
= P(z_n|a_n, S_n)P(x_n|z_n) \int P(a_n|c_n, \beta)P(\beta|\omega) \, \partial \beta \\
\int P(m_n|c_n, \mu, \Sigma)P(\mu|\lambda, \upsilon)P(\Sigma|\phi, \xi) \, \partial \mu, \Sigma,
\]

where the variables \( \hat{\phi}, \hat{\lambda}, \hat{\upsilon} \) and \( \hat{\xi} \) are defined before in Eq. 5.18. It is easy to see the integral with respect to priors \( \int P(a_n, z_n, x_n, m_n, \mu, \Sigma, \beta|c_n, S_n, \Theta) \, \partial H_0(\mu, \Sigma, \beta) \) for unoccupied clusters is equivalent to setting \( o_k = 0 \) in Eq. 5.29, which can be written as

\[
\int P(a_n, z_n, x_n, m_n, \mu, \Sigma, \beta|c_n, S_n, \Theta) \, \partial H_0(\mu, \Sigma, \beta) = P(z_n|a_n, S_n)P(x_n|z_n) \int P(a_n|c_n, \beta)P(\beta|\omega) \, \partial \beta \\
\int P(m_n|c_n, \mu, \Sigma)P(\mu|\lambda, \upsilon)P(\Sigma|\phi, \xi) \, \partial \mu, \Sigma,
\]

By combining the structure of the Bayesian network and the conditional distributions described above, we can write the joint distribution as

\[
P(c, a, z, x, m, \mu, \Sigma, \beta, \lambda, \upsilon, \phi, \xi, \gamma, \omega|S, \theta) = \\
\prod_{n=1}^{N} \left[ P(\gamma|\theta)P(c_n|\gamma)P(\lambda|\theta)P(\upsilon|\theta)P(\phi|\theta)P(\xi|\theta)P(m_n|c_n, \lambda, \upsilon, \phi, \xi) \right. \\
P(\omega|\theta)P(a_n|c_n, \omega)P(z_n|a_n, S_n, \theta)P(x_n|z_n, \theta) \!
\]
where $\theta$ represents the model hyper-parameters for the hyperpriors placed on $\gamma$, $\lambda$, $v$, $\varphi$, $\xi$, and $\omega$.

5.2.2 Inference Method

The combination of complicated interactions of hidden variables and priors in $i$PTMClust leads to a complex joint distribution over high-dimensional spaces which is impossible to characterize analytically in its entirety. This prevents the application of optimization-based inference methods like the EM algorithm. Instead, a common alternative approach is used and it involves approximate inference algorithms such as MCMC sampling (Sec. 3.2.11). In MCMC sampling, the model posterior distribution can be cleverly sampled to collect instances of parameter and variable settings likely under the model. Given a large enough collection of samples, the posterior can be approximated, and the ideal settings of model parameters and hidden variables can be inferred.

Although the Gibbs sampling algorithm is a commonly used MCMC sampling method for nonparametric Bayesian clustering models such as ours, it can mix poorly and produce poor results when the input dataset is large and complex (see Sec. 3.2.11). Hence, in addition to the Gibbs sampling method, we derived and implemented the restricted Gibbs sampling split-merge algorithm (or the split-merge sampling algorithm for short) (Sec. 3.2.11) for $i$PTMClust. The split-merge sampling algorithm is designed to mitigate the aforementioned issues with the Gibbs sampling method. We have applied the following recommended settings for running this sampling algorithm [185]: i) five Gibbs sampling steps after each splitting of clusters, ii) three split-merge steps and iii) a Gibbs sampling step after a set of split-merge steps is completed. We begin by assigning all input peptide sequences to a PTM group and then adhering to the split-merge sampling procedure.

Both the Gibbs sampling split-merge algorithm and Gibbs sampling methods require to perform Gibbs sampling steps. For each Gibbs sampling step, we employ the following
procedure. First, we sample the set of hidden variables (parameters are considered as hidden variables) associated with an input peptide \( n \), where \( n \in [1, ..., N] \). Starting with the \( n \)-th input observation, for each represented (occupied) cluster \( K_{\text{rep}} \), we draw \( a_n \) according to Eq. 5.23 and \( z_n \) using Eq. 5.28. Next, we sample a new cluster assignment \( c_n \) from Eq. 5.1. Finally, we update the sufficient statistics associated with assignments for \( c_n \) and \( a_n \). We repeat this procedure for each input peptide sequence. At the end of each Gibbs sampling step we obtain a new value for each hyper-parameter, with the exception for \( \gamma \), \( \omega \) and \( \xi \), directly by sampling from their conditional posterior distribution given in Eq. 5.13, Eq. 5.13 and Eq. 5.16, which are all distributions of standard form. Although they are not standard distributions, the conditional posteriors for \( \gamma \), \( \omega \) and \( \xi \) are all log-concave, which implies that they are unimodal in the logarithmic domain and have a single global optimum. The log-posteriors of \( \gamma \), \( \omega \) and \( \xi \) are given as

\[
\ln P(\gamma|k, N) = C + \left(k - \frac{3}{2}\right) \ln(\gamma) - \frac{1}{2\gamma} \ln \Gamma(\gamma) - \ln \Gamma(N + \gamma),
\]

\[
\ln P(\omega|A, N) = C + \left(k - \frac{3}{2}\right) \ln(\omega) - \frac{1}{2\omega} \ln \Gamma(\omega) - \ln \Gamma(N + \omega),
\]

\[
\ln P(\xi|\Sigma_1, ..., \Sigma_k, \varphi) = C - k \ln \Gamma \left(\frac{\xi}{2}\right) - \frac{1}{2\xi} + \frac{k\xi - 3}{2} \ln \frac{\xi}{2} + \sum_{k=1}^{K_{\text{rep}}} \frac{\xi}{2} (\ln \Sigma_k + \ln \varphi) - \frac{\xi \Sigma_k \varphi}{2},
\]

where \( C \) is a normalizing constant. Given the equations for the log-posteriors and their log-concave property, a new value for each hyper-parameter can be efficiently sampled using the slice sampling method [198], which is an efficient sampling method and is simple to implement.

Ideally, an unlimited number of MCMC samples should be collected to fully and accurately estimate the posterior. However, in practice, it is common to terminate the sampling procedure after a fixed number of sampling iterations at the cost of posterior estimation accuracy. The number of finite samples to collect is based on an analysis of
how well the samples have mixed by evaluating trace plots of the distribution of posterior probabilities and the number of clusters over time. Furthermore, common to all MCMC methods is a burn-in period at the beginning during which the sampling algorithm mixes poorly. The samples in this burn-in period need to be removed. The number of burn-in sampling iterations to remove from consideration is also determined by examining the trace plots. Despite being data dependent, we found that 1,000 burn-in samples and a total of 15,000 samples are enough to produce a good approximation of the posterior for large-scale PTM datasets, such as the phosphopeptides, yeast proteome and human protein-protein interaction datasets that we used in this study. For simpler data, such as the synthetic and phosphorylation datasets, a setting of 100 burn-ins and 6,000 total samples is sufficient. A detailed analysis is provided later in Sec. 5.3.5. Lastly, to counter auto-correlations amongst the samples, we only use results taken from every fifth sample.

5.2.3 Background Model

Unlike PTMClust, the background model for iPTMClust does not explicitly encompass a predefined background component. Instead, it uses a background model consisting of multiple background components learned directly from the input data. Based on the modification mass variance calculated for each PTM group, we define a background component to be a PTM group with a variance $\geq 2$. This threshold is chosen based on the assumption that the variance for each PTM group is small, because we believe that for a PTM group to be physically relevant, it should have a well-defined modification mass. Therefore, PTM groups with a large modification mass variance are believed to contain spurious data. By allowing for multiple background components instead of one, we observe empirically that the new model is better at capturing spurious data.
5.3 Experimental Evaluations

Our first goal is to demonstrate that $iPTMClust$ outperforms existing algorithms both in terms of finding correct clustering assignments and in refining PTMs on a set of noisy modified peptide sequences. To this end, we conducted two experiments: first, a benchmark of $iPTMClust$ versus standard clustering algorithms, $k$-means and MOG, as well as our foregoing algorithm PTMClust; second, a comparison of blind PTM search engines SIMS, InsPecT and MODmap, state-of-the-art PTM refinement algorithm PTMFinder and our algorithms PTMClust and $iPTMClust$ on detecting the true modification positions from a well-studied phosphopeptide dataset. In each of the experiments, we utilize both the split-merge and the Gibbs sampling inference algorithms for $iPTMClust$. We report results based on MAP estimation for $iPTMClust$ for both experiments. In the second experiment, we explore the advantage of interpreting results with an averaging over samples approach.

The second goal is to directly show the applicability of $iPTMClust$ to datasets taken from studies of complex protein solutions. To achieve this goal, we analyze data taken from a genome-wide yeast and a human chromatin-specific protein complex study. We have limited our analyses to only post-processing PTM predictions generated from SIMS with either PTMClust or $iPTMClust$. We include analysis from PTMClust to highlight that $iPTMClust$ is producing sensible results.

In the experiments, we initialize $k$-means, MOG and PTMClust with settings that are outlined in Sec. 4.3. The settings for the number of burn-in and total samples for $iPTMClust$ are outlined in Sec. 5.2.2.

5.3.1 Algorithm Comparison on Synthetic PTM Data

The dataset used for this analysis is the synthetic dataset outlined in Sec. 4.2 whereas the set of experiments conducted is described in Sec. 4.3.1 where we ran the clustering
algorithms on datasets with varying complexities. This synthetic dataset is designed to simulate situations where multiple PTMs have similar modification masses. The clustering algorithms we compare against are \textit{k}-means clustering, MOG and PTMClust. Instead of using the correction rate (CR) as we have done previously in Sec. 4.3.1, we evaluate our methods using the modified Rand index \cite{199, 200}, which measures the similarity of a set of cluster assignments to the ground truth. The values range from 0 (poor) to 1 (perfect), where a higher value indicates a greater percentage of data are correctly classified as siblings or non-siblings. This modified Rand index provides an alternative quality measure that benefits from not requiring definitions for positives and negatives, and can be written as

\[
R(C, \hat{C}) = \frac{\sum_{i> j} [c_i = c_j \land \hat{c}_i = \hat{c}_j]}{2 \sum_{i> j} \hat{c}_i = \hat{c}_j} + \frac{\sum_{i> j} [c_i \neq c_j \land \hat{c}_i \neq \hat{c}_j]}{2 \sum_{i> j} \hat{c}_i \neq \hat{c}_j},
\]

where \(C\) is the ground truth clustering assignment, \(\hat{C}\) is the one obtained by one of the algorithms and \(c_i (\hat{c}_i)\) indicates the cluster data point \(i\) is assigned to according to \(C (\hat{C})\).

In Fig. 5.2, we show the benchmark result between the surveyed algorithms and \textit{i}PTMClust implemented with both the split-merge and the Gibbs sampling methods. Except for the simplest case where two PTM groups are added, for all other cases both variations of \textit{i}PTMClust generally outperform PTMClust, MOG and \textit{k}-means clustering. When the problem is simple, such as when two PTM groups are added, less complicated algorithms can perform faster and marginally better, as in the case for MOG. The key observation is that \textit{i}PTMClust using the split-merge algorithm attains the most consistent results across different settings and achieves increasingly better results than PTMClust as the problem becomes more complex. Lastly, the issue that Gibbs sampling method can mix poorly is highlighted as more PTM groups are added, e.g., when five PTM groups added, \textit{i}PTMClust with the Gibbs sampling method performs worse than PTMClust. Although not shown, both versions of \textit{i}PTMClust automatically found the correct number of PTM groups in each experiment.
Figure 5.2: Benchmark comparison of clustering algorithms on a synthetic dataset. Rand index is a measure of how closely a list of predicted cluster assignments matches to the ground truth, where 1 indicates a perfect match. The plot shows the new algorithm $iPTMClust$ using the split-merge inference algorithm outperforms all other algorithms as the number of PTM groups in the dataset increases (increasing complexity). More importantly, it performs consistently well while all other algorithms exhibit a drop in performance, especially simpler algorithms like $k$-means and MOG. Furthermore, when the problem is complex, i.e., for 4 and 5 PTM groups added, $iPTMClust$ using the Gibbs sampling performs significantly worst than the one using the split-merge algorithm, which highlights the issues discussed with Gibbs sampling. Due to the simplicity of the problem when two PTM groups are added, MOG is able to perform marginally better than $iPTMClust$ and PTMClust.
Figure 5.3: Evaluation of iPTMClust and PTMClust on a synthetic dataset. (a) shows the error rate as the number of PTM groups increases (increasing complexity) based on the number of correctly identified modified amino acids. (b) shows the error rate based on the number of correctly identified modification positions. Even though in some cases the refined modification position is incorrect, the identified modified amino acid can be correct, which is illustrated by the error rate in (a) is constantly lower than in (b). iPTMClust using the split-merge inference algorithm outperforms (lower error rates) other methods in all experimental settings except when three PTM groups are added, where PTMClust is only marginally better.
Given the true modification positions are known, we can evaluate further how well PTMClust and implementations of our new algorithm perform for the task of PTM refinement. Both K-means and MOG are omitted from this analysis because they do not produce refined modification positions. We define the error rate to be the fraction of peptides that have an incorrect refined modified amino acids. Similarly, we can define the error rate based on the refined modification positions. Given the error rate as a quality measure, Fig. 5.3 shows the same trend as above: iPTMClust with the split-merge algorithm generally outperforms the others. An exception occurs when three PTM groups are added, where PTMClust attains marginally better results than iPTMClust; there are no obvious reasons for this phenomenon. The results from this experiment show that iPTMClust with the split-merge algorithm performs consistently well across a range of settings.

5.3.2 Benchmarking Against Phosphopeptide Predictions

Next, we compare iPTMClust against current PTM search engines and PTM refinement algorithms using a real-world dataset enriched for phosphopeptides containing modification sites that are validated [189]. We will refer to the identities of the known peptide sequences and their modification sites as the reference. This dataset is also used in the previous chapter, and its description can be found in Sec. 4.3.2. The dataset consists of 1655 spectra, but we will focus exclusively on the 1340 spectra mapped and curated as singly modified phosphopeptides (SIMS, InsPecT, PTMClust and iPTMClust are limited to one modification per peptide sequence). In this analysis, we define positives (P) as outputs from the base blind PTM search engine that match to the reference considering only their peptide sequence, i.e., disregarding the positions of their modification, and negatives (N) as all other outputs that do not match their corresponding reference peptide sequences. Each blind PTM search engine produces different number of P and N. For SIMS, PTMClust and iPTMClust (SIMS was used as the base blind PTM search
engine), there are 895 P and 445 N. Lastly, for PTMFinder, which uses InsPecT as its base unrestricted PTM search engine, there are 860 P and 480 N.

For iPTMClust, in addition to the MAP estimate, we considered inference by averaging over samples to produce a confidence score per modification position for each output peptide sequence. By varying the confidence threshold setting, we can adjust the sensitivity and specificity of the PTM predictions. For iPTMClust results evaluated using confidence scores, we define a PTM prediction as a peptide sequence and its modification positions with confidence scores above the threshold; peptide sequences that do not have at least one modification position with a confidence score above the threshold are considered to be assigned to the background model. For PTMClust and iPTMClust using MAP estimate, a prediction is any peptide sequence not assigned to the background model. All peptide sequences assigned to the background model are removed from the evaluation. Lastly, we consider all outputs as predictions for the other algorithms since they do not employ a background model.

Given the true phosphorylation of each peptide are known, we plot the number of correct phosphorylation sites versus the number of incorrect phosphorylation sites identified in Fig. 5.4 for the results of iPTMClust, PTMClust, PTMFinder and InsPecT. For this experiment, the blind PTM search engine InsPecT was used to analyze the input spectra, then each of the PTM refinement algorithms, iPTMClust, PTMClust and PTMFinder, were used to post-process the PTM predictions output from InsPecT. For iPTMClust, we included results for both the split-merge and Gibbs sampling methods using either MAP estimation or inference by averaging over samples. We varied the confidence threshold to obtain a series of results for iPTMClust using inference by averaging over samples. Four key observations can be made from the result: 1) iPTMClust outperforms all other algorithms, including PTMClust and PTMFinder; 2) iPTMClust with inference by averaging over samples produces better results than its MAP estimate counterpart; 3) a background model is essential for collecting spurious modifications; and
Figure 5.4: (a) is a plot of the number of correct phosphorylation sites identified v.s. the number of false phosphorylation sites identified for iPTMClust, PTMClust, InsPecT and PTMFinder on a phosphopeptide dataset with known modification sites. The PTM predictions output from InsPecT are used as the baseline and post-processed by iPTMClust, PTMClust and PTMFinder. The curves for iPTMClust using the split-merge and the Gibbs sampling algorithm are produced by calculating a confidence score per modification position through inference by averaging over samples and varying the confidence score threshold [0,1]. These methods for evaluating the result outperform their counterparts using MAP estimation. More importantly, iPTMClust using any one of the inference methods achieves better results than InsPecT, PTMClust and PTMFinder. (b) is the same plot as (a) zoomed into the region with low number of false phosphorylation sites identified. It highlights a region where iPTMClust using the Gibbs sampling algorithm performs marginally better than its counterpart using the split-merge algorithm.
4) \textit{iPTMClust} with the split-merge inference algorithm performs similarly to \textit{iPTMClust} with the Gibbs sampling algorithm except at the region with low number of incorrect phosphorylation sites identified shown in Fig. 5.4(b), where the Gibbs sampling method is performing marginally better than its counterpart. The first observation reinforces our conclusion from the study on synthetic data that \textit{iPTMClust} outperforms other PTM refinement algorithms. The second highlights the advantage of providing a confidence score per modification position, where adjusting for the confidence threshold allows us to achieve improved results over MAP estimation. Finally, the results highlight a major benefit of \textit{iPTMClust} beyond PTM refinement: i.e., removal of noisy data.

Next, we conducted the same experiment by post-processing the output from SIMS with PTMClust and \textit{iPTMClust} to ensure that our algorithms are not bias towards any one particular blind PTM search engine. Similarly, we plot the number of correct phosphorylation sites versus the number of incorrect phosphorylation sites identified in Fig. 5.5 for the results of \textit{iPTMClust}, PTMClust and SIMS. Since PTMFinder is tightly integrated into the InsPecT algorithm, we were not able to decouple PTMFinder from InsPecT to post-process the output from SIMS. Therefore, PTMFinder is omitted from this analysis. The results based on the PTM predictions generated by both SIMS and InsPecT (discussed above) show the same three trends: 1) \textit{iPTMClust} outperforms all other algorithms; 2) \textit{iPTMClust} with inference by averaging over samples produces better results than its MAP estimate counterpart; and 3) a background model is essential for collecting spurious modifications. Specific to this experiment based on the output from SIMS, we observe that \textit{iPTMClust} with the split-merge inference algorithm markedly performs better than \textit{iPTMClust} with the Gibbs sampling in two conditions: when MAP estimate is used and at the region between 30 to 145 number of false phosphorylation sites identified when using inference by averaging over samples.
Figure 5.5: (a) is a plot of the number of correct phosphorylation sites identified v.s. the number of false phosphorylation sites identified for \( i\text{PTMClust} \), \( \text{PTMClust} \) and SIMS on a phosphopeptide dataset with known modification sites. The PTM predictions output from SIMS are used as the baseline and post-processed by \( i\text{PTMClust} \) and \( \text{PTMClust} \). The curves for \( i\text{PTMClust} \) using the split-merge and the Gibbs sampling algorithm are produced by calculating a confidence score per modification position through inference by averaging over samples and varying the confidence score threshold \([0,1]\). These methods for evaluating the result outperform their counterparts using MAP estimation. More importantly, \( i\text{PTMClust} \) using any one of the inference methods achieves better results than SIMS and \( \text{PTMClust} \). (b) is the same plot as (a) zoomed into the region with low number of false phosphorylation sites identified. It highlights a region where \( i\text{PTMClust} \) using the Gibbs sampling algorithm performs marginally better than its counterpart using the split-merge algorithm.
5.3.3 Large-scale PTM Analysis of Yeast Proteome

Through a series of benchmark experiments, we have shown that iPTMClust beats state-of-the-art algorithms, including our own PTMClust. Furthermore, we demonstrate that iPTMClust using the split-merge sampling method produces improved results over the one using the Gibbs sampling method. Next, we will test iPTMClust’s versatility in detecting diverse PTM groups by applying it to analyze a large-scale PTM dataset taken from analyses of the yeast proteome (LC-MS/MS spectra only) [121] using SIMS. This is the same dataset used previously in Sec. 4.3.3. Briefly, the yeast dataset consists of over 2 million ion trap MS/MS spectra of which 19,560 putatively modified peptides were identified by SIMS with modification range (0, 200] Da. The estimated false discovery rate for the predictions made by SIMS is 4.3% based on the number of decoy peptides identified. Here, we augment our prior reported finding (Table 4.2) with the result taken from our post-analysis using iPTMClust with the split-merge algorithm (MAP estimate was used). MAP estimate is used to simplify the analysis and the mapping to known PTMs. The specific setting used for PTMClust can be found in Sec. 4.3.3.

A summary of commonly known PTMs taken from the Uniprot knowledgebase (Release 2010.11) that are found in our dataset is shown in Table 5.1 and Table 5.2. Overall, iPTMClust using either the split-merge or the Gibbs sampling method is able to reposition a large portion of modifications to known PTM sites that were missed by SIMS originally (increase of ~49% unique PTMs and ~34% modified peptides for split-merge, and ~55% and ~43% for Gibbs). This represents a significant increase over what can be achieved using PTMClust. The most improvement is gained with phosphorylation sites, where post-analysis with iPTMClust is able to identify >65% more (known) unique sites and almost double the number of instances of phosphopeptide when compared to the result obtained from SIMS. However, iPTMClust using the split-merge method incorrectly places a number of peptides with acetylation and other modifications in the background model that SIMS correctly identified. We note that iPTMClust using the
Table 5.1: Summary of known PTM sites in the yeast proteome dataset.

<table>
<thead>
<tr>
<th>PTM</th>
<th>Known PTM Sites (% improvement over SIMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/PTMClust</td>
</tr>
<tr>
<td></td>
<td>Split-merge</td>
</tr>
<tr>
<td>Phosphorylation</td>
<td>103</td>
</tr>
<tr>
<td>Acetylation</td>
<td>5</td>
</tr>
<tr>
<td>Cysteine Oxidation (Cysteine sulfinic acid)</td>
<td>2</td>
</tr>
<tr>
<td>Others</td>
<td>2 (49%)</td>
</tr>
<tr>
<td>Total</td>
<td>112 (~49%)</td>
</tr>
</tbody>
</table>

The known set of modifications was taken from Uniprot (Release 2010_11). We matched the sets of modified peptides produced by SIMS and post-processed with iPTMClust with the split-merge algorithm, iPTMClust with the Gibbs sampling algorithm and PTMClust to the set of known yeast modification sites. The results show iPTMClust is able to improve upon SIMS and significantly outperform PTMClust in a complex dataset.
Table 5.2: Summary of peptides with known PTM sites in the yeast proteome dataset.

<table>
<thead>
<tr>
<th>PTM</th>
<th>i PTMClust Split-merge</th>
<th>i PTMClust Gibbs</th>
<th>PTMClust</th>
<th>SIMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phosphorylation</td>
<td>196</td>
<td>185</td>
<td>115</td>
<td>100</td>
</tr>
<tr>
<td>Acetylation</td>
<td>59</td>
<td>78</td>
<td>75</td>
<td>72</td>
</tr>
<tr>
<td>Cysteine Oxidation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Cysteine sulfinic acid)</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Others</td>
<td>24</td>
<td>35</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>Total</td>
<td>286 (~34%)</td>
<td>304 (~43%)</td>
<td>232 (~9%)</td>
<td>213</td>
</tr>
</tbody>
</table>

Same as Table 5.1, the modified peptides produced by SIMS and post-processed with iPTMClust with the split-merge algorithm, iPTMClust with the Gibbs sampling algorithm and PTMClust against known set of modifications was taken from Uniprot (Release 2010_11). This table shows the number of peptides with known PTMs. Again, the results show iPTMClust is able to improve upon SIMS and significantly outperform PTMClust in a complex dataset.
Gibbs sampling method and PTMClust did not make this mistake. A closer look reveals that many of these instances belong to a few unique peptide sequences. The analysis on the yeast proteome dataset confirms that iPTMClust can detect other PTMs such as acetylation and cysteine oxidation (cysteine sulfinic acid) in addition to phosphorylation. Moreover, the results reiterate that iPTMClust using either the split-merge or the Gibbs sampling method can refine a greater number of PTMs than PTMClust.

### 5.3.4 Analysis of Human Protein-protein Interaction Data

Protein complexes and protein-protein interactions studies are a major focal point in the field of proteomics. However, to date the focus has been mainly on finding complex memberships and interaction partners. Since it is well established that PTMs, such as phosphorylation and acetylation, play a vital role in the formation and regulation of protein-protein interactions, we seek to complement these studies with an emphasis on the identification of PTMs.

The dataset we used consists of high mass-resolution MS/MS spectra (Orbitrap mass spectrometer) from a human protein-protein interaction study searched using SIMS. The study is a collaboration with the Emili lab at the University of Toronto, and the dataset is not yet published at the time of writing. The experimental protocol used is tandem affinity purification (TAP) [201, 202] followed by MS approach. Briefly, the method proceeds by placing a biological tag on all instances of a member of the complex of interest. Next, these tagged proteins are isolated and purified along with their interacting proteins, and finally, the set of purified proteins are subjected to MS analyses.

Here, we chose to focus on three well-studied protein complexes, the Mediator (MED), the RNA Polymerase II (POL2) and the Polycomb Repressive Complex 1 (PRC1). This dataset consists of 17 experiments. These experiments include technical replicates, covering six of 12 proteins known to be in the MED complex, two experiments covering two of the 32 members of the POL2 complex and three experiments covering two of 12
Table 5.3: Listing of unique putative phosphorylated and acetylated peptides for the MED, POL2 and PRC1 complex.

<table>
<thead>
<tr>
<th>Protein Name</th>
<th>Peptide Sequence</th>
<th>Putative PTM</th>
<th>Refined Modification Mass</th>
<th>Refined Modified Amino Acid</th>
<th>Protein Residue of Modified Amino Acid</th>
<th>Number of Instances Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>MED C1QBP_HUMAN</td>
<td>^AFVDFLSDEIK@$</td>
<td>Acetylation</td>
<td>42.0135</td>
<td>K</td>
<td>91</td>
<td>8</td>
</tr>
<tr>
<td>MED C1QBP_HUMAN</td>
<td>^AFVDFLSDEIK@$</td>
<td>Acetylation</td>
<td>42.0124</td>
<td>K</td>
<td>91</td>
<td>14</td>
</tr>
<tr>
<td>MED C1QBP_HUMAN</td>
<td>^M@SSGGWELEINTEAKS</td>
<td>Acetylation</td>
<td>42.0203</td>
<td>M</td>
<td>105</td>
<td>7</td>
</tr>
<tr>
<td>MED C1QBP_HUMAN</td>
<td>^VEEQEPELTSTPNFVVEVIK@$</td>
<td>Acetylation</td>
<td>42.1061</td>
<td>K</td>
<td>174</td>
<td>1</td>
</tr>
<tr>
<td>MED HNRH1_HUMAN</td>
<td>^MLGTEGGEGFVVK$</td>
<td>Acetylation</td>
<td>42.0555</td>
<td>M</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>MED MBD3_HUMAN</td>
<td>^MTQTQPQGAVITTPIPKS</td>
<td>Phosphorylation</td>
<td>80.0562</td>
<td>T</td>
<td>805</td>
<td>12</td>
</tr>
<tr>
<td>MED MED1_HUMAN</td>
<td>^DSTSIDCPAIGT@PLR</td>
<td>Phosphorylation</td>
<td>80.0663</td>
<td>T</td>
<td>1051</td>
<td>12</td>
</tr>
<tr>
<td>MED MED13_HUMAN</td>
<td>^VDFQGCS@PKPESIKS</td>
<td>Phosphorylation</td>
<td>80.0498</td>
<td>S</td>
<td>890</td>
<td>1</td>
</tr>
<tr>
<td>MED MED13_HUMAN</td>
<td>^TQDQVPPPETPS@DSNK</td>
<td>Phosphorylation</td>
<td>80.0489</td>
<td>S</td>
<td>194</td>
<td>1</td>
</tr>
<tr>
<td>MED POL2</td>
<td>^AFVDFLSDEIK@$</td>
<td>Acetylation</td>
<td>42.0345</td>
<td>K</td>
<td>91</td>
<td>2</td>
</tr>
<tr>
<td>MED POL2</td>
<td>^M@ODDEEETRS</td>
<td>Acetylation</td>
<td>42.0469</td>
<td>M</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>MED POL2</td>
<td>^M@HGGGMPGSACPLRS</td>
<td>Acetylation</td>
<td>42.0302</td>
<td>M</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>MED POL2</td>
<td>^M@NPAPFESULFEGEKS</td>
<td>Acetylation</td>
<td>42.0273</td>
<td>M</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MED POL2</td>
<td>^M@NPAPFESULFEGEKS</td>
<td>Acetylation</td>
<td>42.0270</td>
<td>M</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MED POL2</td>
<td>^M@PDEQGTVGFPVGR</td>
<td>Acetylation</td>
<td>42.0190</td>
<td>M</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>MED POL2</td>
<td>^YSPTRPYSPS@PYVTPTPSKS</td>
<td>Phosphorylation</td>
<td>80.0479</td>
<td>S</td>
<td>1899</td>
<td>1</td>
</tr>
<tr>
<td>MED POL2</td>
<td>^YTPTRPYSPS@PYVTPTPSKS</td>
<td>Phosphorylation</td>
<td>80.0560</td>
<td>S</td>
<td>1849</td>
<td>1</td>
</tr>
<tr>
<td>MED POL2</td>
<td>^SFPYS@PNSSTSEVSSTASKS</td>
<td>Phosphorylation</td>
<td>80.0487</td>
<td>S</td>
<td>593</td>
<td>2</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^AFVDFLSDEIK@$</td>
<td>Acetylation</td>
<td>42.0127</td>
<td>K</td>
<td>91</td>
<td>6</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^AFVDFLSDEIK@$</td>
<td>Acetylation</td>
<td>42.2243</td>
<td>M</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^M@SSGGWELEINTEAKS</td>
<td>Acetylation</td>
<td>42.2243</td>
<td>M</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^VEEQEPELTSTPNFVVEVIK@$</td>
<td>Acetylation</td>
<td>42.1061</td>
<td>K</td>
<td>174</td>
<td>1</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^CBXE_JHUMAN</td>
<td>Acetylation</td>
<td>42.0220</td>
<td>M</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^CBXE_JHUMAN</td>
<td>Acetylation</td>
<td>42.0408</td>
<td>M</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^M@LGGTEGGEGFVVK$</td>
<td>Phosphorylation</td>
<td>80.0482</td>
<td>M</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^MSSGGGK@PGSSPITK</td>
<td>Acetylation</td>
<td>42.0299</td>
<td>K</td>
<td>1152</td>
<td>2</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^M@ODDEEETRS</td>
<td>Acetylation</td>
<td>42.0313</td>
<td>M</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^M@HGGGMPGSACPLRS</td>
<td>Acetylation</td>
<td>42.0315</td>
<td>M</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^M@NPAPFESULFEGEKS</td>
<td>Acetylation</td>
<td>42.0319</td>
<td>M</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^M@NPAPFESULFEGEKS</td>
<td>Acetylation</td>
<td>42.0318</td>
<td>M</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^M@PDEQGTVGFPVGR</td>
<td>Acetylation</td>
<td>42.0190</td>
<td>M</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^M@SSGGWELEINTEAKS</td>
<td>Acetylation</td>
<td>42.0299</td>
<td>K</td>
<td>279</td>
<td>1</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^YSPTRPYSPS@PYVTPTPSKS</td>
<td>Phosphorylation</td>
<td>80.0479</td>
<td>S</td>
<td>1899</td>
<td>1</td>
</tr>
<tr>
<td>MED PRC1</td>
<td>^SFPYS@PNSSTSEVSSTASKS</td>
<td>Phosphorylation</td>
<td>80.0487</td>
<td>S</td>
<td>593</td>
<td>2</td>
</tr>
</tbody>
</table>

There are 15 of 48 (~ 31%) that are putative novel modifications that do not map to any known modification sites. Highlighted (bold and italic) entries contained at least one instance that was corrected by iPTMClust in which its modification position was misplaced. The refined modification position is indicated by "@" after the residue. The protein complex label listed indicates the complex associated with the experiment that the peptide was identified.
members of the PRC1 complex. There is a total of 13,221 spectra mapped to modified peptides by SIMS (estimated false discovery rate of 13.5% based on the number of decoy peptides identified) with a modification range \((0, 300]\) Da.

Table 5.3 highlights the 48 distinct peptide sequences putatively identified to be modified either with acetylation or phosphorylation. Including duplicates, a total of 114 modified peptides were found. Of the 48 unique peptides, 9 had at least one instance with their modification site corrected by \(i\)PTMClust using the split-merge sampling algorithm that was originally misplaced by SIMS. These are highlighted in bold. We speculate this smaller number improvement is due to two major reasons: 1) a majority of the listed PTMs are acetylation, which SIMS does a good job with, as seen in the yeast study above, and 2) spectra are cleaner due to the use of a high-resolution mass spectrometer. In this list, there are 15 putative novel modification sites and 33 known ones according to the Uniprot knowledgebase (Release 2011_12). Although the complete list is not shown, after removing those assigned to the background model, we have identified a total of 10,409 putative modified peptides such that a large portion of them is mapped to regions in their respective proteins that do not contain known PTMs. Similar to those listed putative, novel phosphorylated and acetylated peptides, we believe this list also contains many high quality, new PTM discoveries. Hence, this list represents a filtered list of high quality candidates for further investigation. We have shown a PTM prediction pipeline comprises of a blind PTM search engine and \(i\)PTMClust can be fruitful in novel discoveries and should be use routinely.

The use of a high mass-resolution mass spectrometer is expected to reduce errors and potentially remove the need for refinement of measured modification masses. Even so, we noticed many instances where the observed modification mass deviates from our refined modification mass by \(\sim 1\) Da. This modification mass error is believed to be due to the presence of isotopes. Although heuristics can be used to account for these mass shifts, such methods can be error-prone and cannot adapt to unforeseen mass errors. Our
results show that iPTMClust can handle such errors and improve the quality of PTM predictions taken from an analysis of a high mass-resolution mass spectrometer.

5.3.5 Analysis of Number of Samples and Burn-in Period Required

Precise estimation of the posterior distribution requires an infinite number of MCMC samples. Since this is not feasible, users of the algorithm are left to decide on the total number of samples to collect and the size of the burn-in period. The common method to determine the appropriate setting is to analyze statistics collected throughout the sampling process, in the form a trace plot where the values of the statistics are plotted over the number of sampling iterations. In our analysis, we focus on distributions of the log joint probability and the number of non-background PTM groups. Since samples are collected using a probabilistic procedure, the values of the statistics collected vary from iteration to iteration. However, regions in the plot that exhibit large step-like patterns or values that are monotonically decreasing or increasing are examples of signs indicating that the sampling method is still in its initializing burn-in stage. The subtle differences between what is expected and what are signals of poor mixing are unclear, and determination of an optimal setting for the burn-in period and total number of iterations rely heavily on experience.

We have provided the trace plots for our experiments on the phosphopeptide data in Fig. 5.6 and yeast proteome and human protein complex dataset in Fig. 5.7. Fig. 5.6 consists of the trace plots for the log joint probability distribution and the number of non-background PTM groups for iPTMClust using both the split-merge and the Gibbs sampling methods, and Fig. 5.7 contains only those using the split-merge sampling algorithm. The top portion of each subplot in each figure shows the result for the full set of samples, whereas the bottom displays the first 2,000 iterations of the corresponding sampling algorithm. The plots for the phosphopeptide dataset illustrate that both split-
Figure 5.6: Trace plots of the log joint probability distribution and the number of non-background PTM groups across the MCMC samples for the phosphorylation dataset. Both (a) and (b) are trace plots for 
\(iPTMClust\) using the split-merge algorithm. (a) shows the trace plots for the log joint probability distribution, and (b) shows the trace plots for the number of non-background PTM groups. Similarly, (c) and (d) are the corresponding trace plots for 
\(iPTMClust\) using the Gibbs sampling method. The top of each subplot is the trace plot for the complete samples collected and the bottom is expanded to highlight the first 2,000 iterations. From (a) and (b), we can see multiple step-like patterns occurring before 1,000 iterations, which are signals that the sampling method have not reach a steady state. Therefore, this observation suggests that a burn-in period of 1,000 iterations is needed.
Figure 5.7: Trace plots of the log joint probability distribution and the number of non-background PTM groups across the MCMC samples for the yeast proteome and human protein complex datasets. Both (a) and (b) are trace plots for PTMClust using the split-merge algorithm on the yeast proteome dataset. (a) shows the trace plots for the log joint probability distribution, and (b) shows the trace plots for the number of non-background PTM groups. (c) and (d) are the corresponding trace plots for the human protein complex dataset using the split-merge algorithm. The top of each subplot is the trace plot for the complete samples collected and the bottom is expanded to highlight the first 2,000 iterations. The plots highlight that first $\sim 1,000$ sampling iterations exhibit a monotonic increase (both the log-posterior probability and the number of non-background PTM groups) indicating that the sampling algorithm have not reach a steady state. Therefore, this observation suggests that a burn-in period of 1,000 iterations is needed.
merge and Gibbs sampling algorithms mix well after 1,000 iterations. A similar trend emerges in the trace plots for the yeast proteome and human protein complex datasets of our algorithm using the split-merge sampling method (see Fig. 5.7). For both datasets, the plots display a steady increase both in the log joint probability value and the number of non-background PTM groups until \( \sim 1,000 \) iterations or slightly before. Based on these observations, we suggested an initial burn-in of 1,000 iterations and a total number of samples of 15,000 when running iPTMClust. The latter number is to obtain enough samples to estimate the posterior. For all experiments, we suggest taking every fifth sample as a method to avoid auto-correction; a number between 5 to 10 is generally used in MCMC methods. Given the complexity of the yeast and human datasets, we believe this guideline will suit most PTM data.

### 5.4 Discussion and Future Directions

We have established the importance of accurately identifying PTMs and their potential roles in clinical studies such as biomarker discovery and drug development. Although thousands of PTM candidates have been reported using blind PTM search engines [8, 39, 78, 80, 81, 82], these blind PTM search algorithms suffer from mass measurement inaccuracy and uncertainty in predicting modification positions, making the findings error prone. Using our first proposed algorithm, PTMClust, we highlighted the benefit of post-processing PTM predictions using a PTM refinement algorithm. We showed that PTMClust achieves a significantly higher PTM prediction accuracy over two blind PTM search engines as well as over the previous state-of-the-art PTM refinement algorithm, PTMFinder.

Despite significant improvements in PTM prediction, PTMClust is not without its limitations. There are three drawbacks of PTMClust that are of particular interest. First, PTMClust employs a greedy-based, non-automatic model selection algorithm.
An issue general to greedy-based algorithms is that they can perform poorly in certain cases. Second, PTMClust requires manual parameterization on the maximum number of PTM groups. A value for this parameter that is too large or too small can adversely affect the final result. Third, PTMClust does not provide a confidence score per modification position. Cases exist where multiple likely modification positions can be found along a peptide sequence, e.g., a phosphopeptide with multiple serines and threonines. By having a confidence score for each modification position, users can observe the fact that multiple modification sites exist and can better interpret the result.

To address these issues, we introduce \textit{iPTMClust}. \textit{iPTMClust} extends PTMClust by using an infinite mixture model approach that achieves the following three benefits: 1) outperforming PTMClust and other PTM refinement algorithms, 2) providing a fully-automated model selection method without the need for any manual parameterization and 3) offering modification position level confidence scores that users can use to assess the quality of the results and to greater refine their analyses. Through a series of benchmark experiments using both synthetic and real (phosphopeptides and yeast proteome) data, we demonstrated that \textit{iPTMClust} better models the PTM generative process and outperforms PTMClust, PTMFinder and other blind PTM search engines. In addition, we analyzed data generated from a yeast proteome study using \textit{iPTMClust} in which we reported an improvement over the base blind PTM search algorithm SIMS in detecting annotated PTMs. Thousands of putative PTMs were found in this analysis. Moreover, in our in-depth look at PTM predictions for three human protein complexes, MED, POL2 and PRC1, \textit{iPTMClust} we identified numerous validated and putative phosphorylated and acetylated peptides that may be involved in the formation and regulation of protein-protein interactions. Further investigations are warranted, but we believe a number of these putative predictions are valid PTMs and can serve to further our understanding of the complexities involved in protein-protein interactions. Our new algorithm \textit{iPTMClust} is easy to use, achieves overall greater performance than the state of the art, provides
confidence scores at the modification position level that allow for a higher flexibility when evaluating potential PTMs and is designed to be broadly applicable to PTM predictions generated from any blind PTM search engine.

We have established iPTMClust is essential for post-processing current PTM predictions generated by blind PTM search engines given that the PTM predictions suffer from mass measurement inaccuracy and uncertainty in predicting modification positions. However, how applicable is iPTMClust going forward given the rapid advancement in mass spectrometer technology? We explore this question by analyzing data generated from high mass-resolution mass spectrometers, e.g., from an Orbitrap. Although mass accuracy has improved with the use of high mass-resolution mass spectrometers, the presence of isotopes, for example, can result in deviations in observed modification masses. In addition, higher mass accuracy does not necessary equate to errorless modification site determination. We have shown in our analysis of the human protein complex data that problems with mass measurement inaccuracy and uncertainty predicting modification positions continue to exist for data generated from high mass-resolution mass spectrometers, such as an Orbitrap used in the experiment. Our results demonstrate that iPTMClust can improve upon PTM predictions taken from data with high mass accuracy, and continue to be a vital component of a genome-wide PTM study.

In designing iPTMClust, we have developed two different inference algorithms, the split-merge Metropolis-Hastings and the Gibbs sampling algorithm. While the Gibbs sampling method is the standard inference algorithm for Bayesian mixture models and IMMs, the split-merge method is shown to perform better when dealing with complex datasets [185]. Results of our synthetic data experiments also exhibit this trend. iPTMClust using the Gibbs sampling method displays a performance drop while its counterpart using the split-merge sampling algorithm perform consistently well as the data used gets more complex. It is important to note that when there are few PTMs within a small modification mass window, say 1-2 Da (e.g., when two or three PTM groups added
in our experiment with the synthetic data), our algorithm using the Gibbs sampling performs at par with or slightly worse than its counterpart using the split-merge method. Furthermore, iPTMClust using the Gibbs sampling runs faster per iteration (\(~ 50\%\) quicker) than iPTMClust using the split-merge method. Hence, for large datasets, where running time can be overwhelmingly long, we recommend running iPTMClust with the Gibbs sampling method as it provides a good trade-off between quality of the result and computational cost.

Despite outperforming its competitors, iPTMClust has a number of limitations. Similar to PTMClust, it is unable to handle more than one modification per input peptide sequence and PTM groups identified can contain multiple PTMs if their modification masses are similar. The latter problem is less of an issue when working with high mass-resolution data. Moreover, iPTMClust does not consider the underlying spectrum when refining a PTM prediction. The presence of certain peaks in the spectrum can add to support to a residue along the peptide sequence as being the modification position. Its limitations notwithstanding, iPTMClust is shown to outperform both PTMClust and previous state of the art in our benchmark tests using both synthetic and real-world PTM data.

One drawback of sampling inference methods in general is their long computation time, which can be significant for large datasets, e.g., on a computer equipped with two Intel Xeon 3.00 GHz dual core CPUs, it took over four days to run 15,000 iterations using the split-merge sampling method for iPTMClust on the yeast proteome dataset (iPTMClust was ran as a single-threaded application). If a trade-off between faster speedup and loss of accuracy is acceptable, we suggest exploring the use of deterministic Bayesian inference methods such as variational approximation methods [130] and Expectation Propagation (EP) [156]. In particular, EP is shown to perform on par with or slightly better than MCMC sampling methods for simpler, low dimension problems. An important investigation is to determine the type of datasets that are beneficial to use EP
instead of the split-merge method as the inference algorithm for $iPTMClust$. 
Chapter 6

Protein Identification by
Hierarchical Affinity Propagation

In MS, proteins are broken into peptides; their mass spectra are measured and analyzed computationally; and a sets of peptide and PTM predictions are produced. Given these predictions, the next step in a computational MS analysis is to infer the set of the proteins that gave rise to the predictions. However, the task of inferring protein identities that generated a set of noisy spectra is nontrivial. It requires first filtering out false peptide predictions and then assigning the remaining ones to their most likely corresponding proteins. The problem of inferring the protein composition from a set of peptide predictions is compounded by the fact that peptides can map to multiple proteins. Furthermore, for complex mixtures, current state-of-the-art algorithms can only successfully map a small fraction of the mass spectra (<20%) and so researchers are investigating ways of using prior knowledge about protein groups based on similar functionality to improve detection [10, 11].

In this chapter, we will present our novel algorithm to tackle the challenge of protein identification from MS/MS spectra, which we call pro-HAP (protein identification by Hierarchical Affinity Propagation). Unique to our approach is that we cast the problem of
protein identification from MS/MS spectra as a two-layer clustering problem. Intuitively, pro-HAP works by assigning spectra to proteins in the bottom layer and designating proteins to protein clusters corresponding to functional groups in the top layer. We use prior knowledge about protein functions to help form functional groups. We assume proteins that are functionally linked are more likely to be co-expressed. Therefore, when selecting a protein to assign a spectrum to, our method will favour a protein that is functionally related to other proteins identified to be in the sample. The prediction performance of pro-HAP is benchmarked extensively against annotated reference spectra from two separate studies and compared favourably with state-of-the-art methods. Finally, we show that a combination of pro-HAP with SIMS (PTM prediction algorithm) and iPTMClust (PTM refinement algorithm) forms an MS analysis pipeline that is able to reveal numerous novel protein complex members and previously overlooked putative PTMs in an analysis of a human protein-protein study.

6.1 Current State of Computational Protein Identification by MS/MS

Removal of false peptide predictions is a critical first step in protein identification by MS/MS. A simple approach to removing false peptide predictions is by applying a score threshold but this was shown to be inadequate [10, 11, 13, 14]. Even when given a set of high confidence peptide predictions, inferring the underlying proteins remains a challenge [10, 11, 13]. The naïve method, when a reference database is used, e.g., in database search and sequence tag approaches, is to simply take proteins that generated the candidate peptides in the set of high confidence peptide predictions [15]. However, correctly inferring the underlying proteins is nontrivial, because multiple proteins with very similar amino acid sequences exist in the reference database, which can lead to peptides mapping to multiple proteins. Reasons for observing multiple proteins with similar amino
acid sequences include naturally and artificially occurring phenomena such as isoforms due to alternative splicing; proteins from the same protein family; sequencing error at either the DNA or protein level; and redundancy in the reference protein database, i.e., two different entries in the protein database for the same protein. Therefore, there is a need for algorithms that go beyond this naive method.

The latest development in protein identification algorithms focuses on modeling the likelihood of proteins’ presence \[10, 11, 13, 22, 96, 97\]. One such method, ProteinProphet (Sec. 2.6.1), the standard that several new algorithms benchmark against, models the probability of a protein’s presence by combining the likelihood of its supporting peptide predictions while taking into account how unique the peptides are within the reference database. To enhance prediction performance, the idea of incorporating prior knowledge was applied recently in two separate algorithms, MSpresso (Sec. 2.6.2) and MSnet (Sec. 2.6.3). Based on the idea of combining both proteomics and genomics information, MSpresso uses gene expression data from microarray experiments as prior information to post-process and reevaluate protein predictions taken from ProteinProphet. MSpresso outputs a refined confidence score for each input protein prediction. Similar to MSpresso, MSnet is also a protein prediction re-scoring method. The new confidence score is calculated based on a weighted sum of a protein’s initial confidence score and the refined confidence scores from other proteins functionally similar to it. For score recalculation, functional relationship between proteins are taken from the Gene Functional Network (GFN) \[98, 99\], which assigns a similarity value to protein pairs. Protein similarity values between proteins are derived from available annotations, which include Gene Ontology (GO) \[111\], experimental protein-protein interaction data and experimental genetic interaction data.

Accuracy of MSpresso and MSnet when compared against ProteinProphet show a significant improvement with the addition of prior knowledge. However, despite their claims, both MSpresso and MSnet fail to identify the complete list of proteins deemed to
be present in the reference ‘gold standard’ datasets used in their respective benchmark experiment \[10, 11\]. A possible explanation for this is the step-wise greedy approach taken by both MSpresso and MSnet. In the analysis pipeline used by both MSpresso and MSnet, predicting the set of proteins and re-scoring these predictions with prior knowledge are two independent steps that are optimized separately. Since greedy algorithms make locally optimal decisions at each stage, the solutions from these algorithms are not guaranteed to be globally optimal \[77\].

Motivated by MSnet and MSpresso, in this chapter, we propose a new protein identification algorithm, pro-HAP (protein identification by Hierarchical Affinity Propagation), that is also guided by prior knowledge. However, our approach has two unique features: first, we formulate the task of protein identification as a two-layer clustering problem; and second, we incorporate prior knowledge during the protein identification process. In our two-layer clustering framework, spectra are assigned to proteins in the bottom layer and proteins are assigned to protein clusters corresponding to functional groups in the top layer. Similar to MSnet, we use GFN as our prior knowledge on protein functions. The underlying clustering method is a newly developed, general-purpose hierarchical clustering algorithm call Hierarchical Affinity Propagation (HAP) \[16\]. We have designed HAP to efficiently solve hierarchical exemplar-based clustering (EBC) problems using a non-greedy approach. EBC (Sec. 3.2.1) is a special type of clustering algorithms that partitions the data in a way such that each partition is associated with its most prototypical data point (exemplar). Hierarchical EBC is concerned with finding hierarchy of exemplars, where exemplars from the previous (lower) layer are treated as data points to be clustered in the current layer. A greedy approach to hierarchical EBC is one that clusters and optimizes one layer at a time. Greedy-based methods are prone to finding locally optimal but globally suboptimal solutions. On the other hand, a non-greedy approach is where all layers are optimized simultaneously to produce a globally optimal solution. By using HAP as the underlying clustering algorithm, pro-HAP attempts to
find a globally optimal setting of peptide-to-protein mappings that maximizes the scores of peptide-to-protein assignments and protein clusters.

In the following sections, we will introduce pro-HAP for solving protein identification through the use of HAP. To tie HAP to the problem at hand, we will first illustrate how to formulate the protein identification problem as a one-layer facility location problem, which is a slight variant of an exemplar-based clustering problem. We will refer to this one-layer version of the protein identification algorithm as pro-FL. Subsequently, we will extend it to a two-layer hierarchical clustering problem that leverages prior knowledge to improve prediction performance, which we call pro-HAP. To illustrate the effectiveness of HAP, we perform a series of experiments based on synthetic data. Our results show that HAP outperforms other greedy approaches in solving EBC problems. For the task of protein identification, we demonstrate that pro-HAP performs better than, or at par with the state-of-the-art algorithms, ProteinProphet, MSpresso and MSnet, on a yeast proteome and a human protein complex dataset. We conclude our analysis by highlighting an MS analysis pipeline incorporating the PTM prediction algorithm SIMS, our PTM refinement algorithm iPTMClust (see Ch. 5) and pro-HAP. Our study on a human protein-protein dataset shows that we are able to detect novel protein complex members and putative PTMs that may be responsible for regulating interactions between proteins.

6.2 Protein Identification as a Hierarchical Clustering Problem

Given the peptide predictions for a set of spectra, the task of protein identification is to determine a set of proteins that best explain these peptide predictions. This process of assigning peptides to proteins can be naturally seen as an EBC problem, or more specifically, a related problem called facility location (FL). In FL, the task is to assign
each customer (peptide) to a facility (protein) that maximizes 1) the sum of similarities (peptide prediction scores) between peptides and their assigned proteins and 2) the sum of assigned proteins’ prior preferences to be used (negative facility start-up costs). The FL problem is highly related to that of the EBC problem, where the set of facilities (exemplars) is disjoint to the set of input data points (customers). Hence, algorithms that solves EBC can be applied with slight modifications to solve FL problems. By viewing the problem of protein identification as FL, we can take advantage of a number of EBC algorithms.

An appealing property of casting the problem of protein identification as an EBC problem is that we can easily extend it to include prior knowledge. For many experimental studies the biological samples of interest, such as tissue specific or cancer cell line samples, are primarily made up of proteins that are functionally linked, e.g., heart tissue samples are enriched for proteins that are known to be involved in blood circulation and calcium channel regulator activities. Therefore, it is natural to find groupings of proteins in these types of biological samples based on the proteins’ functional annotation. This knowledge can be incorporated into protein identification by EBC to form a two-layer clustering problem, where spectra are assigned to proteins in the bottom layer and proteins are assigned to protein clusters corresponding to functional groups in the top layer. Unique to our approach is that we perform this hierarchical clustering non-greedily, where its attempts to find an optimal solution by considering the clustering assignment at both layers simultaneously. Below, we will illustrate how Affinity Propagation (AP) (Sec. 3.2.8) can be used to solve the problem of protein identification as a one-layer FL problem (pro-FL). Then, we extend the one-layer version to a two-layer version that leverages on prior knowledge by using HAP. This resulting protein identification algorithm is called pro-HAP. Details of the updates to AP to solve FL problems are given in the Appendix C and an extension of AP to HAP (non-greedy hierarchical version of AP) is given in the Appendix D.
6.2.1 Protein Identification by FL

The one-layer FL approach to protein identification, referred to as pro-FL, involves assigning spectra to proteins based on the peptide prediction for each spectrum. Given \( M \) protein sequences of the organism of interest, we can combine the \( N \) input peptide predictions (i.e., spectra-to-peptide mappings) and the mappings of peptides-to-proteins (taken from aligning the \( N \) peptide sequences to the \( M \) protein sequences) to obtain \( L \) spectra-to-protein mappings, where \( L \geq N \). The set of \( M \) proteins can be taken from the reference protein database used during spectral analysis or downloaded from protein knowledgebases such as Uniprot [203]. Based on the spectra-to-protein mappings and the \( N \) input peptide prediction scores (one for each peptide prediction per spectrum), we define the spectra-protein score \( s_{ij} \) for the \( i \)-th spectrum and the \( j \)-th protein as the \( i \)-th peptide prediction score if the \( i \)-th spectrum maps to the \( j \)-th protein or 0 otherwise, where \( i = 1, \ldots, N \) and \( j = 1, \ldots, M \); e.g., if a spectrum is mapped to proteins \( j \) and \( k \) then both spectra-to-protein mappings \( s_{ij} \) and \( s_{ik} \) are set to the corresponding \( i \)-th peptide prediction score. Since spectra-to-protein mappings have a many-to-many relationship, the task of the algorithm, given the inputs, is to assign each spectrum to a protein that maximizes the sum of spectra-protein scores \( s_{ij} \)'s and the protein preferences \( c_j \)'s of chosen proteins, where \( c_j \)'s are the free model parameters in AP (Sec. 3.2.8 for details on AP and Appendix C for modification of AP to solve FL problems). Since in general we do not have any preference for any proteins \textit{a priori}, we have assigned all \( \{c_j\} \)'s to have the same value, and refer to a setting for one \( c_j \) to represent a setting for all \( \{c_j\} \)'s.

An important element for identifying a reliable set of proteins given the inputs is to remove erroneous peptide predictions. We accomplish this by adding a ‘dummy protein’ acting as the background model to collect false peptide predictions. This makes the total number of proteins (facilities) \( M \) to be equal to the number of potential proteins plus one. In addition, we introduce the background preference \( c_b \), which we fixed to have a high preference value to ensure the background model is always present. Given the
inputs and model parameter $c_j$, the algorithm assigns each peptide prediction to one of $M$ proteins, which includes the dummy protein. Hence, the setting for model parameter $c_j$ controls the number of proteins identified and the rate of false predictions, as a smaller value for $c_j$ makes proteins less preferable and more peptides assigned to the background model. By adding decoy proteins into the reference protein database during the spectral analysis (see Sec. 2.2.1), the setting of $c_j$ is chosen by weighting the trade-off between number of real proteins detected and decoy proteins allowed.

### 6.2.2 pro-HAP: Protein Identification with Prior Knowledge

The base one-layer model (pro-FL), described earlier, can be extended to include prior knowledge about the functional relationship between proteins in the sample of interest. Based on the idea that functionally linked proteins tend to be co-expressed, we expect proteins that are identified from samples which have tissue or functional specificity to form protein clusters corresponding to functional groups. This prior knowledge can be incorporated into the one-layer protein identification algorithm by FL described above through the introduction of a second layer. The idea is that proteins identified in the first layer are clustered based on their functional relationships between each other in the second layer. The motivations behind this two-layer clustering design are to allow the protein clusters at the second layer to influence the choice of proteins identified as facilities at the first layer, and vice versa. Therefore, by finding a global optimal solution to this two-layer clustering problem, we will have solve the problem of protein identification guided by prior knowledge.

We use a non-greedy approach to solve the protein identification problem guided by prior knowledge. Our approach is framed as a two-layer clustering method, where the spectra-to-protein assignments and protein clusters are considered simultaneously. To this end, we employ our general propose hierarchical EBC clustering algorithm HAP (see Appendix D) to develop our protein identification by HAP (pro-HAP) algorithm.
The inputs to pro-HAP include $s_{ij}^1$ and $c_{ij}^1$, inputs for pro-FL described above, and two additional inputs: 1) similarity scores $s_{j,o}^2$ between pairs of proteins indexed by $j$ and $o$, where $j = 1, ..., M$ and $o = 1, ..., M$; and 2) exemplar preferences $c_o^2$ that indicate how preferable a protein is to be chosen as an exemplar. The superscripts 1 and 2 indicate that the variables belong to the first and second layer, respectively. Here, we set $s_{j,o}^2$ to be the functional similarity between two proteins, taken from the GFN. Both $c_{ij}^1$ and $c_o^2$ are free model parameters that we adjust manually to achieve the desired solution.

Given the inputs and settings for the model parameters $c_{ij}^1$ and $c_o^2$, pro-HAP estimates the optimal set of proteins that best explain the input peptide predictions.

### 6.3 Experiments and Results

Unique to our proposed protein identification algorithm is the use of our newly developed non-greedy hierarchical EBC algorithm, HAP, to perform protein identification guided by prior knowledge. Below, we first demonstrate that HAP using a non-greedy approach to hierarchical EBC achieves better clustering results than greedy approaches. We conduct two experiments, one using a synthetic 2D dataset and the other using a dataset of synthetic HIV sequences, to show that HAP, majority of the cases, outperforms its greedy bottom-up counterpart (Greedy) as well as a hierarchical version of $k$-medoids and $k$-means clustering using multiple performance measures. Next, we examine the ability of pro-HAP to perform protein identification. Using datasets from a large-scale yeast genome-wide and a human protein-protein interaction study, we benchmark both the one-layer version of our model (pro-FL) and pro-HAP against three state-of-the-art algorithms: ProteinProphet [13], MSpresso [11] and MSnet [10]. Lastly, we highlight an MS analysis pipeline that combines analyses using our PTM refinement algorithm iPTMClust and pro-HAP to further our understanding of protein complex formation and regulation.
6.3.1 Evaluation of HAP on Synthetic 2D Data

As a first step, we conduct experiments on synthetic 2D data to determine whether our proposed method, HAP, obtains better results in terms of the objective function we optimize (also given in Eq. 6.6)

\[
F = \sum_{i,j,l} S_{ij}^l(h_{ij}^l) + \sum_{j,l} C_j^l(e_j^l) + \sum_i I_i^l(h_i^l) + \sum_{i,l>1} I_i^l(h_i^l, e_i^{l-1}) + \sum_{j,l} E_j^l(h_{j,j}^l, e_j^l)
\] (6.1)

when compared to its greedy bottom-up counterpart used in [145], denoted by ‘Greedy’.

To examine the applicability of our method under various settings, we look at varying numbers of points between [100, 750] and layers between [2, 7]. For each such setting we randomly generated increasing exemplar preferences \( c_j^l \) for every layer, and 20 different 2D hierarchical datasets were sampled top-down such that the points of the top-most layer were sampled from a standard normal distribution with a large standard deviation of three. At consequent layers, sampled points from the previous layer were considered the exemplars. Given these exemplars, the layer’s points were sampled by randomly picking one of the exemplars, and sampling a point from a normal distribution centred around the exemplar, with decreasing standard deviation (halved at every layer going down). The total numbers of points sampled at every layer was such that a total overall was of roughly 100, 200, 500, or 750 points across all layers. Also, we ensure from top down every layer contains twice as many points as in the previous layer.

In Fig. 6.1 we report the median percent improvement over the greedy method for every configuration. The median was generated by aggregating over the 20 different datasets for every configuration. Also the figure shows scatter plots of the experiments, where each point represents a result for one of the total of 120 experiments carried out for every choice of the total number of points.

We note that for a larger number of layers (e.g., 5-7 layers, compared to 1-4) the improvement of HAP over the greedy method is more pronounced, with improvements of up to 40% for seven layers and 500 data points. Furthermore, the bar charts and scat-
Figure 6.1: 2D synthetic data: comparison of objective Eq. 6.1 achieved by HAP and its greedy counterpart (Greedy). Top: Median percent improvement of HAP over Greedy for a given number of layers used. Bottom: Scatter plots of the net similarity achieved by HAP v.s. Greedy. Experiments for which HAP obtains better results than Greedy are below the line. Total percent of settings where HAP outperforms Greedy is reported in the inset. Colour in scatter-plot indicates the number of layers.

ter plots show that as dataset size increases, HAP increasingly outperforms the greedy method. Both of these observations are consistent with the expectation that as the hierarchical clustering problem becomes more difficult (more layers and more data points), the gains provided by HAP increase. In a small fraction of cases, HAP does not converge appropriately so that the greedy method achieves better solutions. Thus, in practice, a good strategy might be to run both methods and pick the better solution based on the objective.

For the 2D dataset, we found that HAP tends to oscillate during inference, which prevents convergence. Therefore, we use the following procedure to expedite runtime in the synthetic 2D and the real HIV data: we run the algorithm for 500 iterations, fix the variable assignment for the bottom-most layer that has not been fixed yet, and repeat until all layers are fixed.
6.3.2 Analysis of Synthetic HIV Sequences

The process of HIV evolution can be viewed as one of stochastically generating a hierarchy of exemplars, where each exemplar corresponds to an HIV sequence. The hierarchy represents the evolutionary tree of the HIV sequences and each exemplar represents a parent HIV sequence that gave rise to a set of mutated HIV sequences (data points assigned to the exemplar). Given a population of sequences, a useful task is to infer a tree that describes the evolutionary relationships between the sequences. However, since ground truth labelling is not available for that data, we will explore a synthetically generated dataset. Starting with a single root sequence, we simulated the evolution of HIV for three generations to form a four-layer tree of sequences. First, we randomly picked a four-letter root sequence of length 40. To generate children sequences, we first sampled the number of children from a truncated geometric distribution with mean 10. Then, each child was generated by drawing the number of mutations from a geometric distribution with mean three and mutating the parent sequence in that many randomly drawn positions. This process was repeated until a four-layer hierarchy was obtained, resulting in a total of 867 sequences. Given the generated sequences, pairwise similarities were obtained using the data-generating mutation process and were set to the logarithm of the truncated geometric distribution evaluated at the observed number of mutations between the sequences.

We compare the abilities of HAP, its greedy counterpart (Greedy), a hierarchical version of $k$-medoids clustering (HKMC), and a hierarchical version of $k$-means clustering (HKMeans) on the task of reconstructing the hierarchy. HKMC works by first identifying exemplars at the lowest layer using $k$-medoids clustering [204], and then clustering the resulting exemplars in the second layer. This process is repeated bottom-up until the top layer is reached. For HKMeans, the algorithm finds the lowest layer using $k$-means clustering, and then post-processed the cluster means to find their nearest points.

We apply the different algorithms on a four-layer hierarchy. For HAP and the Greedy,
we vary the exemplar preferences to obtain solutions differing in the number of exemplars at different layers; a total of 304 settings are used. For HKMC and HKMeans, we perform the experiment using a total of 5624 settings generated by varying the number of clusters per layer, including the ground-truth setting. For each setting, we report the result of the best of 100 random restarts using different initializations for the median or cluster centre set.

We evaluate the reconstructed trees using two methods. First, we plot precision vs recall for the following clustering settings: varying exemplar preferences for HAP and Greedy and random initialization for HKMC and HKMeans. We label each sequence-layer combination as positive if the sequence was generated in that layer in the ground truth, and as negative otherwise. The precision vs recall for different methods are plotted in Fig. 6.2. These results demonstrate that HAP outperforms the other methods at nearly all recall levels. Note that this is also true for the case where HKMC and HKMeans are given the correct parameters for the number of clusters at each layer as input.

Additionally, the recall level at which HAP ‘breaks’ and performance converges to random performance is significantly higher (\(\sim 0.82\)) than that of the other three methods (Greedy \(\sim 0.67\), HKMC & HKMeans \(\sim 0.62\)). Note that applying standard AP layer-by-layer (Greedy) outperforms HKMC, which is expected since, for the single-layer case, AP was shown to outperform \(k\)-medians clustering \[166\].

For a more in-depth comparison of HAP and Greedy, we applied an additional evaluation method using modified Rand index (previously used in Sec. 5.3.1), which measures how frequently pairs of data points are correctly classified as siblings or non-siblings by the hierarchical clustering algorithms, given the ground truth clustering. Fig. 6.3 compares HAP and Greedy for different parameter settings by plotting the Rand index values for each layer in the ground truth labelling\[1\]. In the high Rand index regime (\(>0.8\)), HAP

---

\[1\] This direct comparison on a per-parameter setting basis cannot be replicated for HKMC and HKMeans since they operate on different parameters, namely, the number of clusters to find.
Figure 6.2: Synthetic HIV data: precision-recall for HAP, Greedy, HKMC and HKMeans applied to the problem of identifying ancestral sequences from a set of 867 synthetic HIV sequences. For HKMC and HKMeans, we only plot the best precision obtained for each unique recall value.
Figure 6.3: Synthetic HIV data: distribution of Rand index for different experiments using HAP and Greedy. A higher Rand index indicates the solution better resembles the ground truth. Experiments for which HAP obtains better results than Greedy are below the line. The percentage of solutions that identified the correct single ancestor sequence at the top layer (layer 4) is also reported.
almost always performs better than Greedy. It is interesting that by considering all layers, HAP is able to achieve significantly higher Rand indices for the bottom layer, as shown by the black triangles. The mean Rand index across all layers and experiments is 0.90 for HAP vs 0.67 for Greedy. For the first, second and third layers, the mean Rand index is 0.91 for HAP vs 0.67 for Greedy, 0.76 for HAP vs 0.62 for Greedy, and 0.86 for HAP vs 0.64 for Greedy. Since the correct solution has a single ancestral exemplar at the top layer, we calculated the fraction of experiments that correctly identifies that single ancestor. In this endeavour, HAP was successful in 84% of the cases, whereas Greedy was successful in only 35% of the cases.

6.3.3 Benchmark Analysis of Yeast Proteome

Having shown that the non-greedy approach to hierarchical EBC triumph greedy approaches, we next focus on the task of protein identification. We begin by examining the benchmark dataset used by both MSpresso and MSnet. This dataset consists of a collection of mass spectra of yeast wild-type grown in rich medium analyzed by LCQ and Orbitrap mass spectrometers [205, 10]. The spectra were further analyzed using Sequest [5] and scored with PeptideProphet (Sec. 2.2.5), which produced more than 223,000 high confidence peptide predictions mapping to \( \sim 6700 \) proteins [5, 9, 10, 205]. This list of proteins contained both real and decoy proteins. Protein cluster scores were extracted from the yeast GFN, which provides similarity scores between pairs of proteins based on various sources of data. Both the peptide prediction scores and protein cluster scores were normalized to the interval \((0, 1]\) and then log-transformed. After normalization, the only free parameters were the protein preferences \( c_j^1 \), which we varied from \(-16\) to \(-1\), and the protein cluster preferences \( c_o^2 \), varied from \(-5\) to 0.

We compared our method to three state-of-the-art algorithms: ProteinProphet, MSpresso and MSnet. We used a curated list of proteins from MSnet as a gold standard and compared the ROC curve of our method to the ROC curves of the other methods.
Figure 6.4: A plot of the true positive rate (TPR) vs false positive rate (FPR) for pro-HAP, three state-of-the-art protein identification algorithms, and an one-layer facility location-based formulation of the problem (pro-FL), on the problem of protein identification using mass spectrometry data derived from yeast wild-type grown in a rich medium. Both pro-HAP and pro-FL outperform ProteinProphet and MSpresso while being on par with MSnet. Interestingly, pro-HAP and pro-FL perform similarly which suggest the use of prior knowledge have little effect. A possible explanation could be the lack of functional specificity in the sample examined, whole-cell yeast lysate.
We applied the settings outlined in [10] for MSnet and ProteinProphet, and in [11] for MSpresso. The gene expression data needed by MSpresso was downloaded from the reference given in [11]. Across a wide range of false positive rates, Fig. 6.4 shows pro-HAP achieves significantly higher true positive rates than ProteinProphet and MSpresso, and performs similarly to the best known method, MSnet. However, when compared to the one-layer reformulation of the problem (pro-FL), pro-HAP did not show significant improvement. This suggests that the inclusion of prior knowledge has a minimal effect. A possible explanation could be the lack of functional specificity in the sample examined, whole-cell yeast lysate. To explore the question about the benefits of prior knowledge in protein identification, we next analyze a dataset from a human protein-protein interaction study. Due to the nature of the study, the samples examined in the human protein-protein interaction study have high functional specificity.

6.3.4 Benchmark Analysis of Human Protein-protein Interaction Data

Tandem affinity purification (TAP) followed by MS analyses is one of the common protocols used to study protein-protein interactions and protein complexes [201, 202]. In this pipeline, proteins of interest are chemically tagged, isolated along with their interacting partners, and then this set of proteins are subjected to MS analyses. Therefore, proteins found in each TAP experiment should have high functional specificity. We make use of this fact in our analysis to examine whether prior knowledge helps improve the predictive performance of pro-HAP.

We analyze a dataset taken from a high mass-resolution MS human protein-protein interaction study and focus on spectra generated for three well-studied protein complexes, the Mediator (MED), the RNA Polymerase II (POL2) and the Polycomb Repressive Complex 1 (PRC1). This is the same set of spectra used in Sec. 5.3.4 and is not yet published at the time of writing. The dataset consists of 17 experiments covering 7 members of the
Figure 6.5: Human Protein-protein Interaction Data: distribution of fraction of protein complex members found to decoy protein detected. The plot shows pro-HAP and pro-FL outperform MSnet and on par with ProteinProphet. Due to the small fraction of spectra mapping to decoy proteins (∼0.17%), both pro-HAP and pro-FL are able to isolate and remove these false predictions.

Three complexes for a total of 38,824 spectra mapped to peptides by Xtandem (estimated false discovery rate of ∼0.17% based on the number of decoy peptides identified). For both ProteinProphet and MSnet, we followed the recommended settings for running ProteinProphet given in [10] and varied the parameter $(1-\gamma)$ in MSnet (Sec. 2.6.3) from 5 to 10, as suggested by Ramakrishnan et al. for human datasets [10]. Similar to benchmark experiments with the yeast proteome dataset, we normalized and log-transformed both the peptide prediction scores and protein cluster scores, and varied the free parameters protein preferences $c_j^1$ from −16 to 0 and the protein cluster preferences $c_o^2$ from −5 to 0.
Given that the identities of proteins in this dataset are unknown, we assume all known protein members of the three complexes are present and treat them as positives. Furthermore, we consider all decoy proteins as negatives. Using these definitions of positives and negatives, in Fig. 6.6 we plot the fraction of known protein complex members vs fraction of decoy proteins found for both pro-HAP and pro-FL, as well as for the two state-of-the-art algorithms, MSnet (with \( \frac{(1-\gamma)}{\gamma} = 5 \)) and ProteinProphet. MSpresso was omitted from this analysis because we do not have corresponding microarray data required by MSpresso. The figure shows that pro-HAP and pro-FL outperform MSnet slightly and are at par with ProteinProphet. Moreover, MSnet performs worse than ProteinProphet. Although not shown, MSnet performs progressively worse as its model parameter \( \frac{(1-\gamma)}{\gamma} \) goes from 5 to 10; a higher value for \( \frac{(1-\gamma)}{\gamma} \) indicates more reliance on the information given in the prior knowledge. We believe the reason MSnet did worst in this experiment is because of its limitation outlined earlier in Sec. 2.6.3: MSnet assumes a complete protein functional network. For human, protein annotations are far from fully elucidated; hence, the GFN for human is incomplete.

Interestingly, both pro-HAP and pro-FL produce results with 0 decoy proteins. Due to the small number of spectra mapping to decoy proteins, both pro-HAP and pro-FL are able to remove these false predictions by assigning them into the dummy protein (background model) across the explored parameter settings. However, the question about how useful prior knowledge is to the problem of protein identification could not be answered by just this result. Next, we attempt to address this question by exploring the same set of spectra analyzed by Sequest, which reported result with a higher number of decoy proteins.

The spectra from the studies of the three protein complexes analyzed by Sequest yield 32,549 confident peptide predictions (estimated false discovery rate of \( \sim 6\% \) based on the number of decoy peptides identified). The plot in Fig. 6.6 shows the comparison between pro-HAP and pro-FL. Both pro-HAP and pro-FL perform very well on eliminating false
Figure 6.6: Human Protein-protein Interaction Data: distribution of fraction of protein complex members found to decoy protein detected for pro-HAP and pro-FL. The spectra in the dataset was searched with Sequest. The plot shows pro-HAP marginally outperform pro-FL, which suggest that the inclusion of prior knowledge has a positive although minor effect.
peptide detections. pro-HAP achieves a slightly better result at the lower regime of fraction of decoy proteins found compared to pro-FL. However, this slight performance gain is not significant enough to suggest the inclusion of prior knowledge has a definite positive impact on quality of prediction. One explanation for the low improvement achieved by the inclusion of prior knowledge is that the dataset examined still had very low false detection, which can be attributed to the use of high mass-resolution mass spectrometer. Although the benefits of using prior knowledge are not clear, we demonstrate that pro-HAP and pro-FL can perform on par or better than state-of-the-art algorithms.

6.3.5 In-depth Analysis of Human Protein-protein Interaction Data

To fully understand the intricacy behind protein complexes, identifying protein complex memberships and interacting partners is simply not enough. Questions such as what are the mechanisms that regulate the formation of physical interactions between proteins cannot be answered. In this section, we propose an MS analysis pipeline that incorporates PTM prediction, PTM refinement and protein identification. The aim is to enhance our insights into the formation and regulation of protein complexes. We accomplish that by integrating information about proteins that are predicted to be involved in the protein complex of interest and PTMs that occur while the said proteins are interacting with the complex. Using our analyses of the dataset for three protein complexes MED, POL2 and PRC1 as an example, we highlight the benefits of this unified analysis approach.

In Table 6.1 we highlight proteins that are identified in our analysis above (spectra analyzed using Sequest) (see Sec. 6.3.4) and are predicted to be either acetylated or phosphorylated (see Sec. 5.3.4 for list of modified proteins). Entries that contain at least one putative novel PTM are highlighted in bold. Overall, using the following setting for pro-HAP model parameters, $c_j^1 = -0.0001$ and $c_o^2 = -0.01$, we identified a total of 882 proteins (356 specific to MED, 363 for POL2 and 163 for PRC1) and mapped
Table 6.1: Table of selected identified proteins with associated putative PTMs.

<table>
<thead>
<tr>
<th>Putative Novel Complex Member</th>
<th>Protein Complex</th>
<th>Protein Name</th>
<th>Number of Instances</th>
<th>Putative PTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>MED</td>
<td>HNRH1_HUMAN</td>
<td>129</td>
<td>Acetylation</td>
</tr>
<tr>
<td></td>
<td>MED</td>
<td>MED1_HUMAN</td>
<td>624</td>
<td>Phosphorylation</td>
</tr>
<tr>
<td></td>
<td>MED</td>
<td>MED13_HUMAN</td>
<td>249</td>
<td>Phosphorylation</td>
</tr>
<tr>
<td></td>
<td>MED</td>
<td>MED19_HUMAN</td>
<td>78</td>
<td>Phosphorylation</td>
</tr>
<tr>
<td></td>
<td>MED</td>
<td>MED23_HUMAN</td>
<td>23</td>
<td>Acetylation</td>
</tr>
<tr>
<td></td>
<td>MED</td>
<td>RPB1_HUMAN</td>
<td>401</td>
<td>Acetylation</td>
</tr>
<tr>
<td></td>
<td>POL2</td>
<td>RPAB1_HUMAN</td>
<td>13</td>
<td>Acetylation</td>
</tr>
<tr>
<td></td>
<td>POL2</td>
<td>RPB1_HUMAN</td>
<td>127</td>
<td>Acetylation, Phosphorylation</td>
</tr>
<tr>
<td></td>
<td>POL2</td>
<td>RPB11_HUMAN</td>
<td>8</td>
<td>Acetylation</td>
</tr>
<tr>
<td></td>
<td>POL2</td>
<td>RPB9_HUMAN</td>
<td>9</td>
<td>Acetylation</td>
</tr>
<tr>
<td>*</td>
<td>POL2</td>
<td>RPRD2_HUMAN</td>
<td>42</td>
<td>Phosphorylation</td>
</tr>
<tr>
<td></td>
<td>PRC1</td>
<td>CBX2_HUMAN</td>
<td>12</td>
<td>Acetylation</td>
</tr>
<tr>
<td></td>
<td>PRC1</td>
<td>CBX4_HUMAN</td>
<td>16</td>
<td>Acetylation</td>
</tr>
<tr>
<td>*</td>
<td>PRC1</td>
<td>CBX6_HUMAN</td>
<td>8</td>
<td>Acetylation</td>
</tr>
<tr>
<td>*</td>
<td>PRC1</td>
<td>HNRH1_HUMAN</td>
<td>4</td>
<td>Acetylation</td>
</tr>
<tr>
<td>*</td>
<td>PRC1</td>
<td>ICLN_HUMAN</td>
<td>1</td>
<td>Phosphorylation</td>
</tr>
<tr>
<td>*</td>
<td>PRC1</td>
<td>KDM2B_HUMAN</td>
<td>27</td>
<td>Phosphorylation</td>
</tr>
<tr>
<td>*</td>
<td>PRC1</td>
<td>PCGF6_HUMAN</td>
<td>11</td>
<td>Acetylation</td>
</tr>
<tr>
<td></td>
<td>PRC1</td>
<td>SCMH1_HUMAN</td>
<td>9</td>
<td>Acetylation</td>
</tr>
</tbody>
</table>

The set of PTMs are taken from Table 5.3. Entries that are found to have at least one putative novel PTM are highlighted in bold.
them to the list of acetylated and phosphorylated proteins given in Table 5.3. We note that all identified proteins are not likely to be truly interacting with their respective complex as this list may contain contaminants and false predictions. By using a combination of protein complex membership and PTM information, we are able to quickly identify possible high-yield novel discoveries that are worthy of further investigation. For example, RPRD2_HUMAN is known to regulate nuclear pre-mRNA domain-containing proteins [203] and is only recently and independently validated to be an interacting partner of members of the POL2 complex [206]. Unique to our analysis and not previously known is the detection of activity of the phosphorylation site at the 593rd residue in RPRD2_HUMAN at the time of the interaction between the protein and the POL2 complex. Given the experimental conditions, we believe the putative novel phosphorylation site for RPRD2_HUMAN that we predicted might play a role in regulating the interaction between RPRD2_HUMAN and the POL2 complex. Beyond RPRD2_HUMAN, we believe there are a number of noteworthy candidates embedded in this table that warrant further analysis, such as KDM2B_HUMAN where a putative novel phosphorylation site at the 912th residue and multiple peptides of the protein are detected in the experiment. This result shows a glimpse of the potential impact our proposed analysis pipeline can have on protein complex-related and more generally proteomic studies.

6.4 Summary and Future Directions

Identifying an optimal pipeline of algorithms for spectral analysis is vital for qualitative and quantitative proteomics studies, including biomarker and protein complex discoveries. In this chapter, we introduced a novel method for protein identification, which uses a two-layer clustering approach to leverage prior knowledge about protein function. This is done by assigning spectra to proteins in the lower layer and clustering functionally linked proteins guided by prior knowledge in the upper layer. Our method uses a
newly developed, general-purpose hierarchical EBC algorithm, HAP. We tested our protein identification algorithm, pro-HAP, and an one-layer version of it, pro-FL, on a yeast dataset and showed that they perform better than state-of-the-art protein prediction algorithms, MSpresso, and ProteinProphet and at par with MSnet. Surprisingly, our initial study shows that comparable results can be achieved with or without the addition of prior knowledge, which is in contrast to the claim in [10]. A possible explanation for this could be the lack of functional specificity in the sample examined, whole-cell yeast lysate. To further investigate the impact of prior knowledge on the task of protein identification, we examined a dataset taken from a human protein-protein interaction study, which consists of the proteins that are highly functionally linked to each other. Again, the results show that pro-HAP and pro-FL perform at par or better than state-of-the-art algorithms. Moreover, the result shows pro-HAP performs only slightly better than pro-FL, which failed to conclusively suggest that the inclusion of prior knowledge is beneficial to the prediction performance. Overall, we concluded that pro-HAP and pro-FL perform at par with or better than the state-of-the-art protein identification algorithms but found minimal effect of prior knowledge on our algorithm.

Beyond simple protein identification, we also proposed an integration of PTM and protein predictions into an MS analysis pipeline that allows a more in-depth analysis of protein complex formation and regulation. The result of this analysis identifies a number of confident potential protein complex members for the MED, POL2 and PRC1 complexes and associated PTMs (specifically acetylation and phosphorylation) for these putative novel members. As an example, we highlighted RPRD2_HUMAN as a potential interacting candidate for the POL2 complex and reveal an active phosphorylation site at the time of the interaction that may play a vital role in RPRD2_HUMAN interacting with the POL2 complex. Our proposed analysis pipeline provides a glimpse of the possible ways that we can mine MS datasets such as protein-protein interaction data.

Despite offering comparable performance to the state-of-the-art algorithms, our algo-
rithm has three possible extensions which we believe are noteworthy. First, we believe an investigation of other forms of prior knowledge is needed. The use of GFN as the prior knowledge was motivated since we had a platform to compare HAP with MSnet. Although GFN attempts to incorporate information from multiple sources such as GO, published studies on protein-protein and genetic interactions, and curated protein functions, it is static and has infrequent releases that makes the data stale. Alternatively, GeneMANIA \[100\] can be used in place of GFN as the prior knowledge. GeneMANIA is a protein function prediction algorithm that also integrates multiple sources of information, and has the following two advantages: 1) the information is provided in real-time using up-to-date information and 2) it is available for multiple organisms, instead of just yeast and human as in GFN. Second, determining a good set of parameters requires re-running our algorithm with various settings. Further investigations into the problem of explicit parameter setting optimization is an ongoing research direction. Lastly, theoretically the loopy brief propagation inference method used in HAP has no guarantee of convergence, which means the solution from our algorithm might not be optimal. This is a longstanding problem in ML. However, we like to note that this is a minor issue thus far as our results empirically show the our algorithm performs quite well.
Chapter 7

Concluding Remarks

In this thesis, we have addressed two key areas in an MS/MS computational analysis: PTM refinement and protein identification. A common theme in our solutions is clustering. Clustering allows us to leverage the statistical power drawn from considering the dataset as a whole instead of each datum individually to answer specific questions of interest. This approach is intuitively similar to the idea of meta-analysis in statistics.

For the problem of PTM refinement, we illustrate in our algorithms, PTMClust (Ch. 4) and its extension \(i\)PTMClust (Ch. 5), that modelling modifications at the PTM level instead of at the peptide level has profound advantages. Similarly, in our protein identification algorithm, pro-HAP (Ch. 6), we show that it is beneficial to view the assignment of spectra to protein guided by prior knowledge as a hierarchical clustering problem.

The central idea in PTMClust and \(i\)PTMClust - refining observed modifications by taking a PTM level approach - is markedly different from the current state-of-the-art PTM prediction and refinement algorithms. The experiments in Sec. 4.3 and Sec. 5.3 show that both PTMClust and \(i\)PTMClust are adept at detecting PTM groups that correspond to previously-annotated, naturally occurring and artificially induced PTMs, and putative novel PTMs. Moreover, based on the PTM groupings, PTMClust and \(i\)PTMClust are able to fine-tune the modification mass and modification position of
each input modified peptide. Using benchmark experiments against state-of-the-art algorithms, we demonstrate that both PTMClust and iPTMClust achieve vastly superior results. While PTMClust already outperforms the state of the art, iPTMClust outshines it with better performance and ease of use, and provides additional beneficial features such as a confidence score per modification site.

The state-of-the-art protein identification algorithms, i.e., ProteinProphet, MSnet and MSpresso, are scoring-based methods. In contrast, we have taken the approach of formulating the problem of protein identification as a two-layer hierarchical EBC problem that exploits prior knowledge about protein function. The benchmarks in Sec. 6.3 show that our one-layer approach without prior knowledge (pro-FL) and the two-layer version (pro-HAP) are at par with or better than the best performing state-of-the-art algorithm in each of our experiments. Surprisingly, our results indicate that prior knowledge has a minimal effect on the prediction performance between pro-FL and pro-HAP, which is in sharp contrast to the claims made in [10, 11]. While our results do not demonstrate that pro-HAP can significantly outperform state-of-the-art methods, they indicate that pro-HAP is consistently ranked amongst the best performers across a wide variety of datasets.

Individually, each of the three proposed algorithms, PTMClust, iPTMClust and pro-HAP, compared favourably to the state-of-the-art algorithms in their respective domain, and have led to the identification of many confident putative PTM and protein predictions, respectively. However, even more significant gains can be achieved by combining the methods into an MS/MS analysis pipeline. We show in Sec. 6.3.5 that such an analysis pipeline can filter for, and selectively yield high confidence, novel protein complex members for a large-scale human protein complex study. Furthermore, by overlaying putative PTM discoveries onto the set of identified proteins, we are able to provide insights into possible factors that may regulate the corresponding protein-protein interactions. Our analysis gives a glimpse of potential uses of such an MS/MS analysis pipeline.
Despite the positive results that each of our three algorithms achieved, we have discussed their limitations and proposed a number of future directions in respective chapters. Below, we outline a few other noteworthy prospective research directions more general to the studies of MS/MS computational analysis and ML.

First, missing is a discussion of drawbacks with the overall, end-to-end MS/MS computational analysis workflow. Current MS/MS computational analysis approaches involve applying a collection of tools or algorithms to solve each of the various stages of the workflow. This can take the form of a unified suite of tools, such as the OpenMS Proteomics Pipeline (TOPP) [207], MaxQuant [208] and the Trans-Proteomic Pipeline (TPP) [209, 210]. Common to these approaches is the fact that each tool is run independently and sequentially, which makes the whole process a greedy-based approach to MS/MS computational analysis. As we have shown in our analysis of synthetically generated data in Sec. 6.3, a non-greedy method can avoid locally optimal but globally suboptimal solutions that can often occur with greedy approaches. Even though the complexity of the problem represents a major challenge, further research in a fully integrated, all-encompassing MS/MS computational analysis algorithm may prove to be fruitful.

Another area worth exploring is to incorporate project-specific expert knowledge into the model in a semi-supervised manner. Project-specific expert knowledge may come in the form of known PTMs, e.g., phosphorylation and acetylation, for PTM refinement and known proteins in the sample for protein identification. This information is currently being used to validate the results but not as input to the models. Semi-supervised ML models like those applied in areas such as computer linguistics [211] and computational biology [212, 213] can be of use here.

Lastly, further improvements can be made in regard to the ML algorithms and techniques used in this thesis. One notable example is the derivation of an efficient version of HAP using the sum-product inference algorithm. While the max-product algorithm offers
many desirable properties such as efficiency for HAP, a sum-product version of HAP can be useful when MAP estimates yield poor solutions. However, two issues arise with the straightforward sum-product version of AP derived in [166]: 1) the update equations cannot be calculated in $O(N^2)$ time complexity due to numerical precision issues and 2) the algorithm is no longer invariant to arbitrary scaling of the input similarities. The same issues will affect a similarly derived sum-product version of HAP. A sum-product-like inference algorithm that avoids the aforementioned drawbacks of sum-product is worth exploring as it can potentially benefit other complex graphical models in addition to AP and HAP.
Appendix A

Derivation of a Mixture of Gaussians (MOG) Model

The following is the derivation for the E- and M-steps in the EM algorithm for MOG. Given a joint distribution $P(x_n, z_n|\theta)$ for $n = 1, ..., N$ data points over observed variables $x_n$ and hidden variables $z_n$ governed by parameters $\theta$, the goal is to maximize the likelihood function $P(x_n|\theta)$ with respect to $\theta$. In a mixture of $K$ Gaussians model, each Gaussian distribution has the form

$$
\mathcal{N}(x|\mu_k, \Sigma_k) = \sum_{n=1}^{N} \frac{1}{\sqrt{2\pi \Sigma_k}} \exp\left\{ -\frac{1}{2\Sigma_k} (x_n - \mu_k)^2 \right\}.
$$

(A.1)

Then, the conditional distribution of $x$ given a particular value of $z$ is

$$
P(x|z = k) = \mathcal{N}(x|\mu_k, \Sigma_k),
$$

(A.2)

where $k = 1, ..., K$. Furthermore, the likelihood function is

$$
P(x, z|\theta) = P(z)P(x|z),
$$

(A.3)

$$
= \alpha \mathcal{N}(x|\mu, \Sigma),
$$
where mixing coefficients $\alpha = P(z)$ satisfy $\alpha \geq 0$ and $\sum_{k=1}^{K} \alpha_k = 1$. From this we can obtain the expected log likelihood, given as

$$L(\theta) = E_z \left[ \ln \prod_{n=1}^{N} P(x_n, z|\theta) \right]$$

(A.4)

$$= E_z \left[ \sum_{n=1}^{N} \ln P(x_n, z|\theta) \right]$$

$$= \sum_{n=1}^{N} E_z [\ln P(x_n, z|\theta)]$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} P(z_n = k|x_n, \theta) \ln P(x_n, z_n = k|\theta)$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} P(z_n = k|x_n, \mu_k, \Sigma_k) \ln [\alpha_k N(x_n|\mu_k, \Sigma_k)]$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} P(z_n = k|x_n, \mu_k, \Sigma_k) \left[ \ln \alpha_k - \frac{1}{2} \ln(2\pi) - \frac{1}{2} \ln(\Sigma_k) - \frac{(x_n - \mu_k)^2}{2\Sigma_k} \right].$$

In the E-step, the posterior distribution (responsibility) $Q$ is calculated using the data points and the current setting of the parameters, given by

$$Q_{nk} = P(z_n = k|x_n, \mu_k, \Sigma_k)$$

(A.5)

$$= \frac{P(z_n = k)P(x_n|z_n = k, \mu_k, \Sigma_k)}{\sum_{j=1}^{K} P(z_n = j)P(x_n|z_n = j, \mu_j, \Sigma_j)}$$

$$= \frac{\alpha_{nk} N(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \alpha_{nj} N(x_n|\mu_j, \Sigma_j)}.$$

In the M-step, while fixing $Q$, the ML setting of the parameters is calculated by setting the derivatives of $\ln P(x|\mu, \Sigma)$ in Eq. (A.4) with respect to each parameter to zero.
and solving for it. For mean $\mu_k$, we have

$$0 = \frac{\partial L(\theta)}{\partial \mu_k}$$

$$0 = \frac{\partial}{\partial \mu_k} \sum_{n=1}^{N} \sum_{k=1}^{K} Q_{nk} \left[ \ln \alpha_k - \frac{1}{2} \ln(2\pi\Sigma_k) - \frac{(x_n - \mu_k)^2}{2\Sigma_k} \right]$$

$$0 = \sum_{n=1}^{N} Q_{nk} \frac{1}{\Sigma_k} (x_n - \mu_k)$$

$$\mu_k = \frac{\sum_{n=1}^{N} Q_{nk} x_n}{\sum_{n=1}^{N} Q_{nk}}.$$  \hspace{1cm} (A.6)

Similarly, for covariance $\Sigma_k$, we have

$$0 = \frac{\partial L(\theta)}{\partial \Sigma_k}$$

$$0 = \frac{\partial}{\partial \Sigma_k} \sum_{n=1}^{N} \sum_{k=1}^{K} Q_{nk} \left[ \ln \alpha_k - \frac{1}{2} \ln(2\pi) - \frac{1}{2} \ln(\Sigma_k) - \frac{(x_n - \mu_k)^2}{2\Sigma_k} \right]$$

$$0 = \sum_{n=1}^{N} Q_{nk} \left[ -\frac{1}{2\Sigma_k} + \frac{(x_n - \mu_k)^2}{2(\Sigma_k)^2} \right]$$

$$\sum_{n=1}^{N} Q_{nk} \left[ \frac{1}{\Sigma_k} \right] = \sum_{n=1}^{N} Q_{nk} \left[ \frac{(x_n - \mu_k)^2}{2(\Sigma_k)^2} \right]$$

$$\Sigma_k = \frac{\sum_{n=1}^{N} Q_{nk} (x_n - \mu_k)^2}{\sum_{n=1}^{N} Q_{nk}}.$$  \hspace{1cm} (A.7)

Finally, for mixing coefficient $\alpha_k$, we add a Lagrangian term to Eq. (A.4) to account for the constrain $\sum_{k=1}^{K} \alpha_k = 1$. Now, to solve for $\alpha_k$ we get

$$0 = \frac{\partial L(\theta)}{\partial \alpha_k}$$

$$0 = \frac{\partial}{\partial \alpha_k} \left\{ \sum_{n=1}^{N} \sum_{k=1}^{K} Q_{nk} \left[ \ln \alpha_k - \frac{1}{2} \ln(2\pi) - \frac{1}{2} \ln(\Sigma_k) - \frac{(x_n - \mu_k)^2}{2\Sigma_k} \right] \right\} - \lambda \left( \sum_{j=1}^{K} \alpha_j - 1 \right)$$

$$0 = \sum_{n=1}^{N} Q_{nk} \left[ \frac{1}{\alpha_k} \right] - \lambda$$

$$\alpha_k = \frac{1}{\lambda} \sum_{n=1}^{N} Q_{nk}.$$  \hspace{1cm} (A.8)
If we sum over $k$ on both side we get

$$
\sum_{k=1}^{K} \alpha_k = \sum_{k=1}^{K} \frac{1}{\lambda} \sum_{n=1}^{N} Q_{nk}
$$

$$
\lambda = \sum_{k=1}^{K} \sum_{n=1}^{N} Q_{nk} \quad \text{(A.9)}
$$

Inserting $\lambda$ from Eq. (A.9) into Eq. (A.8) we get

$$
\alpha_k = \frac{\sum_{n=1}^{N} Q_{nk}}{\sum_{n=1}^{N} \sum_{k=1}^{K} Q_{nk}} \quad \text{(A.10)}
$$
Appendix B

Derivation of PTMClust

Below, we derive the EM algorithm for PTMClust to infer the unobserved variables and learn our model parameters, which in this case include the PTM group probabilities $\alpha_k$, the modification mass means $\mu_k$, modification mass variances $\Sigma_k$, and the probability that the PTM occurs on an amino acid $\beta_{ki}$ for each PTM group.

Using the notation given in Sec. 4.2, we derive the steps in EM for PTMClust as follows. To avoid numerical underflow all the calculations are performed in the logarithmic domain. In the E-step, the log joint probability for iteration $t$ is calculated as follows:

$$\ln P(c, a, z, x, m|S, \theta_t^{-1}) = \ln P(c, a, z, x, m|S, \alpha_t^{-1}, \mu_t^{-1}, \Sigma_t^{-1}, \beta_t^{-1})$$ (B.1)

where $\theta_t^{-1} = \{\alpha_t^{-1}, \mu_t^{-1}, \Sigma_t^{-1}, \beta_t^{-1}\}$ represents the set of parameters at iteration $t - 1$.

From Eq. 3.13, we have the responsibility $Q$ as

$$Q(t)(c, a, z) = P(c, a, z|x, m, S, \theta_t^{-1})$$ (B.2)

$$= \frac{\sum_{n=1}^{N} P(c_n, a_n, z_n, x_n, m_n|S, \theta_t^{-1})}{\sum_{n=1}^{N} \sum_{c} \sum_{a} \sum_{z} P(c_n, a_n, z_n, x_n, m_n|S, \theta_t^{-1})}.$$ (B.2)

Since all calculations are done in logarithmic domain, we will need to obtain the posterior distribution $P(c, a, z|m, x, S, \theta_t^{-1})$ from the log-likelihood (this is known as the log-sum trick):
\[ P(c, a, z | x, m, S, \theta^t) = \]

\[ \frac{\sum_{n=1}^{N} \alpha_k}{\sum_{n=1}^{N} \sum_{c} \sum_{a} \sum_{z} \exp \left[ \ln P(c_n, a_n, z_n, x_n, m_n | S, \theta^t) - \max_{c, a, z} \left( \ln P(c_n, a_n, z_n, x_n, m_n | S, \theta^{t-1}) \right) \right]} \]

In the M-step, the parameters are estimated by maximizing the expected complete log-likelihood. We begin by writing down the expected complete log-likelihood as follows:

\[ \langle \ln P(c, a, z, x, m | S, \theta^t) \rangle_{Q^t} = \int_{\theta^t} \ln \prod_{n=1}^{N} P(c_n, a_n, z_n, x_n, m_n | S, \theta^t) \]

\[ = \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{i=1}^{A} \sum_{j=0}^{L_n} Q^{(t)}(c_n = k, a_n = i, z_n = j) \ln P(c_n = k, a_n = i, z_n = j, x_n, m_n | S, \theta^t) \]

To account for the constraints \( \sum_{k=1}^{K} \alpha_k = 1 \) and \( \sum_{i=1}^{A} \beta_{ki} = 1 \), we add one Lagrangian term per constraint to the expected complete log-likelihood and expanding it to give

\[ \langle \ln P(c, a, z, x, m) \rangle_{Q^{(t)}} = \left[ \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{i=1}^{A} \sum_{j=0}^{L_n} Q^{(t)}(c_n = k, a_n = i, z_n = j) \right] \ln(N) + \ln(\alpha_k) + \ln(\beta_{ki}) - \ln(\delta_i + 1)[S_n(j) = i][j > 0] - \frac{1}{2} \ln(2\pi \Sigma_k) - \frac{(m_n - \mu_k)^2}{2\Sigma_k} + \phi(x_n - j)[j > 0] + \phi(L_n)[j = 0] \] - \lambda \left[ \sum_{k=1}^{K} \alpha_k - 1 \right] - \tau \left[ \sum_{i=1}^{A} \beta_{ki} - 1 \right] \]

To obtain a new estimate of each parameter, we set the derivative of the expected complete log-likelihood Eq. [B.5] with respect to each of the parameters to zero and solve for the corresponding parameter. Setting the derivative of the expected complete log-
likelihood with respect to the means $\mu_k$ of the PTM groups to zero, we obtain

$$0 = \frac{d}{d\mu_k} \langle \log P(c, a, z, x, m|\theta^t) \rangle_{Q(t)}$$

$$0 = -\sum_{n=1}^{N} \sum_{i=1}^{A} \sum_{j=0}^{L_n} Q(t)(c_n = k, a_n = i, z_n = j) \frac{1}{\Sigma_k} [m_n - \mu_k]$$

$$0 = -\sum_{n=1}^{N} Q(t)(c_n = k) \frac{1}{\Sigma_k} [m_n - \mu_k].$$

Multiplying by $\Sigma_k$ and rearranging to solve for $\mu_k$, we obtain

$$\mu_k = \frac{\sum_{n=1}^{N} Q(t)(c_n = k)m_n}{\sum_{n=1}^{N} Q(t)(c_n = k)}.$$

(B.6)

In the above expression, $Q(t)(c_n = k)$ is computed from $\sum_i \sum_j Q(t)(c_n = k, a_n = i, z_n = j)$.

Similar to the derivation of $\mu_k$ in Eq. B.6, we set the derivative of the expected complete log likelihood with respect to the variances $\Sigma_k$ to zero and following a similar line of reasoning to solve for $\Sigma_k$, we obtain

$$\Sigma_k = \frac{\sum_{n=1}^{N} Q(t)(c_n = k)(m_n - \mu_k)^2}{\sum_{n=1}^{N} Q(t)(c_n = k)}.$$

(B.7)

Next, we calculate a new estimate for the mixing coefficients $\alpha_k$. The constraint that mixing coefficients sum to one ($\sum_{k=1}^{K} \alpha_k = 1$) is enforced by the addition of a Lagrangian term in Eq. B.5. Maximizing the expected complete log likelihood with respect to $\alpha_k$, we obtain

$$0 = \sum_{n=1}^{N} \sum_{i=1}^{A} \sum_{j=0}^{L_n} Q(t)(c_n = k, a_n = i, z_n = j) \frac{1}{\alpha_k} - \lambda$$

$$0 = \sum_{n=1}^{N} Q(t)(c_n = k) \frac{1}{\alpha_k} - \lambda.$$

Multiplying the above equation by $\alpha_k$ and summing over $k$, making use of the constraint $\sum_{k=1}^{K} \alpha_k = 1$, we get $\lambda = N$. Using this to eliminate $\lambda$ and rearranging, we obtain

$$\alpha_k = \frac{1}{N} \sum_{n=1}^{N} Q(t)(c_n = k).$$

(B.8)
Finally, maximizing the expected complete log likelihood with respect to $\beta_{ki}$, which also consists of a Lagrangian term to account for the constraint $\sum_{i=1}^{A} \beta_{ki} = 1$, we obtain

$$0 = \sum_{n=1}^{N} \sum_{j=0}^{L_n} Q(t)(c_n = k, a_n = i, z_n = j) \frac{1}{\beta_{ki}} - \tau$$

$$0 = \sum_{n=1}^{N} Q(t)(c_n = k, a_n = i) \frac{1}{\beta_{ki}} - \tau.$$ 

Using the same procedure as above to eliminate $\tau$ and rearranging, we get

$$\beta_{ki} = \frac{\sum_{n=1}^{N} Q(t)(c_n = k, a_n = i)}{\sum_{n=1}^{N} Q(t)(c_n = k)}.$$ 

(B.9)

At the end of each pair of E- and M-steps, we calculate the log-likelihood and check for convergence. If the convergence criterion (the difference between current and previous expected log-likelihood is smaller than $10^{-5}$) is not satisfied, we cycle through to the next iteration and repeat both the E- and M-steps.
Appendix C

Extending AP to Solve FL Problems

The goal a facility location (FL) problem is to find a subset of facilities to utilize, and assign each customer to a facility. FL is very much like EBC with one major difference, the set of facilities (exemplars) is disjointed from the input data points. Like EBC, FL is NP-hard, and like EBC, AP can be used to obtain good quality, albeit approximate, solutions. The extension of AP to solve FL problems was previously proposed in [214]. Below, we highlight the modification to AP that allow AP to solve an FL problem. We will make use of the formulation and notation of AP given in Sec. 3.2.8.

To account for the fact that facilities (exemplars) are disjointed from the input data points, there are two sets of changes to the AP model. First, we are assigning $N$ data points to $M$ facilities. This means, as inputs to the algorithm, we are given the similarities $s_{ij}$ between data point $i$ and facility $j$ and a set of facility preference $\{c_j\}$ and $c_j$ is the preference for opening facility $j$ as one of the facilities, where $i = 1, ..., N$, $j = 1, ..., M$. Also, there are $M$ copies for factor nodes $\{E_j\}$ and $\{C_j\}$. Second, the constraint equation Eq. 3.37 is replaced by the following:

$$E_j(h_{ij}, e_j) = \begin{cases} 
0 & \text{if } e_j \geq h_{ij} \ \forall i, \\
-\infty & \text{otherwise.}
\end{cases}$$

where the difference between Eq. C.1 and Eq. 3.37 is the omission of the constraint that
an exemplar (facility), if chosen by a data point, must also choose itself as an exemplar (i.e., \( e_j = h_{jj} \) if \( e_j \geq h_{ij} \ \forall i \)). The definition for functions \( C_j, S_{ij} \) and \( I_i \) remain the same as in AP and are restated as follows:

\[
C_j(e_j) = c_je_j, \quad (C.2)
\]
\[
S_{ij}(h_{ij}) = s_{ij}h_{ij}, \quad (C.3)
\]
\[
I_i(h_i) = \begin{cases} 
0 & \text{if } \sum_j h_{ij} = 1, \\
-\infty & \text{otherwise.} \end{cases} \quad (C.4)
\]

Lastly, the objective function remains the same as in AP and is given by:

\[
\mathcal{F}({\{h_{ij}\}},{\{e_j}\}) = \sum_{i,j} S_{ij}(h_{ij}) + \sum_j C_j(e_j) + \sum_i I_i(h_i) + \sum_j E_j(h_{ij}, e_j). \quad (C.5)
\]

Same as in AP, finding an approximate MAP solution to this NP-hard problem is achieved by running the max-sum algorithm. The message updates in the algorithm are derived similar to those in AP. They are simplified to the following two messages, termed availability \( (\alpha_{ij}) \) and responsibility \( (\rho_{ij}) \), that are calculated iteratively until convergence:

\[
\alpha_{ij} = \min[0, -c_j + \sum_{k\neq j} \max(0, \rho_{kj})], \quad (C.6)
\]
\[
\rho_{ij} = -s_{ij} - \max_{k\neq j}(\alpha_{ik} - s_{ik}). \quad (C.7)
\]

The difference between AP and FL is the computation of \( \alpha_{ij} \), where in FL the condition to check if \( i = j \) is omitted from Eq. 3.39 to account for the change in \( E_j \) described above.
Appendix D

Derivation of HAP from AP

We now show how to generalize the flat AP model to a hierarchical one. We wish to find a set of $L$ consecutive layers of clustering, where the points to be clustered in layer $l$ are constrained to be in the exemplar set of layer $l-1$. This guarantees a hierarchy of exemplars, and as we move up in the layers, an exemplar either remains an exemplar or chooses another exemplar to be its exemplar, relinquishing its own role as a cluster representative. Since a greedy, layer by layer solution may incur lower preferences for exemplar choices at higher layers due to locally optimal but globally suboptimal decisions made at lower layers, we wish to find a global solution.

Fig. D.1(a) describes a graphical model for the hierarchical EBC problem. The main
Figure D.1: (a) HAP factor-graph, a single layer of the standard AP model is shown in the dotted square. (b) HAP messages.
difference compared to the flat representation is manifested in the $I^l_i$ functions as follow:

\begin{align*}
C^l_j(e^l_j) &= c^l_j e^l_j, \quad (D.1) \\
S^l_{ij}(h^l_{ij}) &= s^l_{ij} h^l_{ij}, \quad (D.2) \\
I^l_i(h^l_{ii}) &= \begin{cases} 
0 & \sum_j h^l_{ij} = 1, \\
-\infty & \text{otherwise.} 
\end{cases} \quad (D.3) \\
I^{l>1}_i(h^l_{ii}, e^{l-1}_i) &= \begin{cases} 
0 & \sum_j h^l_{ij} = e^{l-1}_i, \\
-\infty & \text{otherwise.} 
\end{cases} \quad (D.4) \\
E^l_{ij}(h^l_{ij}, e^l_j) &= \begin{cases} 
0 & e^l_j = h^l_{jj} = \max_i h^l_{ij}, \\
-\infty & \text{otherwise.} 
\end{cases} \quad (D.5)
\end{align*}

The modified $I^{l>1}_i$ function results in the following behaviour: if point $i$ is not chosen as an exemplar at layer $l - 1$, (i.e. if $e^{l-1}_i = 0$), then point $i$ will not be clustered at layer $l$. Alternatively, if point $i$ is chosen as an exemplar at layer $l - 1$, it must choose an exemplar at layer $l$. Although we do not explicitly require that only exemplars from previous layers serve as exemplars at the current layer, this will implicitly be guaranteed due to both the fact that non-exemplar points in layer $l$ will be constrained to not choose an exemplar, and the exemplar consistency constraint that simply allows a point to be an exemplar if it has chosen itself.

The second difference, manifested through the $S^l_{ij}$ and $C^l_j$ functions, is that we allow for layer-specific pairwise similarities ($s^l_{ij}$) and exemplar preferences ($c^l_j$). The rest of the functions remains as before, with the appropriate layer superscripts. As is the case for standard AP, there are no restrictions on the form of the input pairwise similarities and preferences. Thus, the objective function we wish to maximize for HAP can be stated as

\begin{align*}
\sum_{i,j,l} S^l_{ij}(h^l_{ij}) + \sum_{j,l} C^l_j(e^l_j) + \sum_i I^l_i(h^l_{ii}) \\
+ \sum_{i,l>1} I^{l>1}_i(h^l_{ii}, e^{l-1}_i) + \sum_{j,l} E^l_{ij}(h^l_{ij}, e^l_j) \quad (D.6)
\end{align*}

As is the case for standard AP, there are no restrictions on the form of the input pairwise
distances. Furthermore, it is possible to define different pairwise distances and exemplar costs for different layers of the hierarchy.

### D.1 Inference by Message Passing for HAP

Similar to AP, we find solutions to the problem we described above by inferring approximate MAP values for the hidden variables using the max-sum algorithm. Since the function nodes between layers and within layers are high-order, naively, it may seem that they require the evaluation of an exponential number of settings when calculating the maximization needed to compute the max-sum function-to-variable messages. However, much like the case of single layer AP, the set of valid settings for the maximization is restricted and a careful analysis of these valid settings, and shared computations allows us to compute function-to-variable messages much more efficiently, resulting in runtime of $\mathcal{O}(LN^2)$ per iteration, where $L$ is the number of layers and $N$ is the number of input data points.

The messages to be passed are shown in Fig. [D.1(b)]. For an AP layer $l$ using the definition of $I^{l>1}_i$ given in Eq. [D.4], the messages $\tau_j^{l+1}$ passed up to the next layer from variable node $e^l_j$ to function node $I^{l+1}_j$ and the message $\phi_j^{l-1}$ passed down to the previous layer from function node $I^l_j$ to variable node $e^{l-1}_j$ are

$$
\tau_j^{l+1} = c^l_j + \rho^l_{jj} + \sum_{k \neq j} \max(0, \rho^l_{kj}) \quad (D.7)
$$

$$
\phi_j^{l-1} = \max_k (\alpha^l_{jk} + s^l_{jk}) \quad (D.8)
$$

The modified $\alpha^l_{ij}$ and $\rho^l_{ij}$ messages for an AP layer are

$$
\alpha^l_{ij} = \begin{cases} 
\hat{c}^l_j + \sum_{k \neq j} \max(0, \rho^l_{kj}) & i = j \\
\min[0, \hat{c}^l_j + \rho^l_{ij} + \sum_{k \neq \{i,j\}} \max(0, \rho^l_{kj})] & i \neq j 
\end{cases} \quad (D.9)
$$

$$
\rho^l_{ij} = s^l_{ij} + \min_{k \neq j} [\tau^l_i - \max_{k \neq k} (\alpha^l_{ik} + s^l_{ik})] \quad (D.10)
$$
We note the \( \rho_{ij} \) messages passed in the first layer and the \( \alpha_{ij} \) messages passed in the top-most layer are identical to the standard AP messages for an AP layer.
Appendix E

SIMS Algorithm

Sequential Interval Motif Search (SIMS) \[8\] is a database search engine that allows for blind modification searches of large collection of MS/MS spectra. It differs from other database search algorithms, such as Sequest \[5\] and Mascot \[6\], in that it performs blind PTM searches without prior knowledge of the specific modification masses and the amino acids in which the PTMs occur on. Details on MS/MS experiments, MS/MS spectra and various types of spectral analysis algorithms are given in Sec. 2.1.

SIMS is broken into three main stages: 1) an initialization or preprocessing stage; 2) a database search stage; and 3) an approval stage. In addition, two parameter files are needed by SIMS: Sequest parameter file (sequest.param), which provides the molecular weight for each amino acid and location of the reference database; and SIMS modification parameter file (simsmod.cfg), which lists the range of allowable PTM modification masses. Next, we discuss each stage of the algorithm in more detail.

Inputs to the preprocessing stage are the two parameter files and an experimental MS/MS spectrum. First, given these inputs, the preprocessing stage identifies all pairs of b- and y-ion peaks, and generates ‘ghost’ b- or y-ion peaks to account for missing complementary b- or y-ion peaks. Second, it assigns a score for each b-ion peak (real and ‘ghost’) based on the significance of the peak to the interpretation of the spectrum.
This peak score is influenced positively by high peak intensity and the presence of nearby peaks due to isotopic effects, and negatively if the peak itself is a ghost peak. Third, it simplifies and denoises the spectrum by binning the selected peaks into one mass unit and picking the highest intensity peak per bin. Finally, the final set of selected peaks are converted into inter-peak intervals, where each interval must correspond to the mass of an amino acid residue.

In the database search stage, guided by amino acid sequences taken from a reference protein database, the algorithm identifies a list of potential peptide sequences that can explain the spectrum. Starting at each possible peptide cleavage location for a reference protein (e.g. lysine or arginine for the enzyme trypsin), each amino acid in the peptide sequence is matched to a peak interval sequentially, allowing for PTMs, in the form a mass shift, until the combined mass of the amino acids and any PTMs reaches the precursor mass. The precursor mass, measured in the first MS scan, is the mass of the peptide that generated the spectrum. Gaps in the predicted amino acid sequence are allowed if corresponding peaks are not found in the list of peak intervals. A score for each matched peak is added, with a negative score given to gaps, to produce a final score for each potential peptide. Only the top 400 scoring potential peptides are kept for the approval stage.

The approval stage validates each of the top scoring peptides identified in the database search stage. This is done by generating a theoretical spectrum for each potential peptide and cross correlating it with the experimental spectrum. For an efficient calculation, both the theoretical spectra and experimental spectrum are Fourier transformed and cross-correlated to the frequency domain. In the case of modified peptides, a theoretical spectrum is generated for each possible PTM site localization. The resulting peptide predictions are ranked according to their cross correlation score.
Bibliography


[80] Liu, C., Yan, B., Song, Y., Xu, Y., Cai, L.: Peptide sequence tag-based blind

S.: Identification of protein modifications using ms/ms de novo sequencing and the

[82] Han, Y., Ma, B., Zhang, K.: Spider: software for protein identification from
sequence tags with de novo sequencing error. Journal of bioinformatics and computa-
tional biology 3(3) (2005) 697–716


[84] Searle, B.C., Turner, M., Nesvizhskii, A.I.: Improving sensitivity by probabilis-
tically combining results from multiple ms/ms search methodologies. Proteome

Hunt, D.F.: The utility of etd mass spectrometry in proteomic analysis. Biochim

[86] Tanner, S., Payne, S., Dasari, S., Wilmarth, P., David, L., Loomis, W., Briggs,
S., Bafna, V.: Accurate annotation of peptide modifications through unrestrictive

[87] Olsen, J.V., Blagoev, B., Gnad, F., Macek, B., Kumar, C., Mortensen, P., Mann,
M.: Global, in vivo, and site-specific phosphorylation dynamics in signaling net-


Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D., Issel-
Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M.,
Rubin, G.M., Sherlock, G.: Gene ontology: tool for the unification of biology: the

robust speech recognition using probabilistic models for clean speech and for noise.
In: European Conference on Speech Communication and Technology (Eurospeech).
(2001)


[116] Frey, B.J., Morris, Q.D., Hughes, T.R.: Genrate: a generative model that re-
veals novel transcripts in genome-tiling microarray data. Journal of computational


[118] Engelhardt, B., Jordan, M., Muratore, K., Brenner, S.: Protein molecular function


[124] Shai, O., Morris, Q.D., Blencowe, B.J., Frey, B.J.: Inferring global levels of alternative splicing isoforms using a generative model of microarray data. Bioinformatics
BIBLIOGRAPHY


[185] Jain, S., Neal, R.: A split-merge markov chain monte carlo procedure for the
Dirichlet process mixture model. Journal of Computational and Graphical Statistics

[186] Hastings, W.: Monte carlo sampling methods using markov chains and their ap-

implications for cancer detection, prevention, and therapeutics. Molecular Cell

Algorithm and computational tools for tandem mass spectrometry based identifica-
tion of post-translational modifications in proteins. Journal of Proteome Research
4 (2005) 2338–2347

[189] Beausoleil, S., Jedrychowski, M., Schwartz, D., Elias, J., Villen, J., Li, J., Cohn,
M., Cantley, L., Gygi, S.: Large-scale characterization of hela cell nuclear phospho-
proteins. Proceedings of the National Academy of Science of United States of

[190] Kim, S., Na, S., Sim, J.W., Park, H., Jeong, J., Kim, K., Seo, Y., Seo, J., Lee,
K.J., Paek, E.: Modi: a powerful and convenient web server for identifying multiple
post-translational peptide modifications from tandem mass spectra. Nucleic Acids

[191] Na, S., Paek, E.: Prediction of novel modifications by unrestrictive search of

[192] Craig, R., Beavis, R.C.: A method for reducing the time required to match protein
sequences with tandem mass spectra. Rapid Communications in Mass Spectrometry
17 (2003) 2310–2316


