Development and Evaluation of Methods for Predicting Protein Levels from Tandem Mass Spectrometry Data

by

Han Liu

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

Abstract

Development and Evaluation of Methods for Predicting Protein Levels from Tandem Mass Spectrometry Data

Han Liu

Master of Science

Graduate Department of Computer Science

University of Toronto

2005

This work addresses a central problem of Proteomics: estimating the amounts of each of the thousands of proteins in a cell culture or tissue sample. Although laboratory methods involving isotopes have been developed for this problem, we seek a simpler approach, one that uses more-straightforward laboratory procedures. Specifically, our aim is to use data-mining techniques to infer protein levels from the relatively cheap and abundant data available from high-throughput tandem mass spectrometry (MS/MS). In this thesis, we develop and evaluate several techniques for tackling this problem. Specifically, we develop and evaluate different statistical models of MS/MS data. In addition, to evaluate their biological relevance, we test each method on three real-world datasets generated by MS/MS experiments performed on various tissue samples taken from Mouse.

Acknowledgements

I would like to express my sincere thanks to my supervisor, professor Anthony Bonner. I feel so honor to have the chance to work with him, he brought me into the fantastic fields of statistical data mining and computational biology. All the works appeared in this thesis are the results from his insight about problems and the remarkable knowledge in algebra and statistics. I also respect his insistence on the preciseness and the rigor about research works as well as scientific writings. I believe that I will benefit from his dedicated supervision greatly in the future academic career.

I should also thank professor Rafal Kustra, the second reader of this thesis, for his time and effort on reviewing this thesis. I want to thank professor Andrew Emily at Banting and Best department of medical research for collaboration with us.

Finally, I would like to dedicate this thesis to my parents, without their selfless support, it is impossible for me to finish this thesis.

Contents

Chapter 1

Introduction

Proteomics is the large-scale study of the thousands of proteins in a cell [32]. In a typical Proteomics experiment, the goal might be to compare the proteins present in a certain tissue under different conditions. For instance, a biologist might want to study cancer by comparing the proteins in a cancerous liver to the proteins in a healthy liver. Modern mass spectrometry makes this possible by enabling the identification of thousands of proteins in a complex mixture [46, 15]. However, *identifying* proteins is only part of the story. It is also important to *quantify* them, that is, to estimate how much of each protein is present in a cell [1, 22]. To this end, a number of laboratory methods have been developed, notably those based on mass tagging with isotopes [23, 39]. However, simpler, more-direct methods may be possible, methods that do not require additional laboratory procedures, but which are simply based on the data provided by tandem mass spectrometers [33]. This paper is an initial exploration of this possibility. In particular, we investigate the use of data-mining techniques to infer protein quantity from tandem mass spectrometry data.

Tandem mass spectrometry (MS/MS) of peptides is a central technology of Proteomics, enabling the identification of thousands of peptides and proteins from a complex mixture [41, 35, 1]. In a typical experiment, thousands of proteins from a tissue sample

are fragmented into tens of thousands of peptides, which are then fractionated, ionized and passed through a tandem mass spectrometer. The result is a collection of spectra, one for each protein, where each peak in a spectrum represents the spectral count of a single peptide [30, 43]. With the increasing acquisition rate of tandem mass spectrometers, there is an increasing potential to solve important biological problems by applying data-mining and machine-learning techniques to MS/MS data [21, 15]. These problems include (*i*) estimating the levels of the thousands of proteins in a tissue sample, (*ii*) predicting the spectral counts of the peptides in a mass spectrum [21], and (*iii*) explaining why different peptides from the same protein have different spectral counts [15]. In this work, we divide these problems into two categories, problems (*i*) and (*ii*) belong to the first category, which is trying to quantitatively predicate the spectral counts and protein levels; while problem (*iii*) belongs to the other category, which is trying to explain the underlying principles of the mass spectrometry data.

In this work, for the second categorical problems, as a first step in explaining why different peptides produce spectral counts of different levels, we reduce the problem to its simplest terms: explaining why some peptides produce higher spectral counts, while others produce lower spectral counts. Moreover, we treat this as a classification problem. That is, given the amino-acid sequence of a peptide, we attempt to classify the peptide as either high-spectral count or low-spectral count. If we can do this reliably, then we have effectively discovered what it is about the amino-acid sequence of a peptide that determines whether it produces high- or low-spectral counts. This is especially true if the parameters of the classifier can be interpreted biologically. To tackle this classification problem, we evaluates and compares a variety of classification methods on MS/MS data. Conducting the evaluation involves three main steps: producing a set of labeled data, training a number of different classifiers on a portion of the data (the "training data"), and evaluating the effectiveness of the classifiers on the remaining data (the "testing data").

For the problems in the first category, we develop and test a number of regression methods for tackling these problems, including methods based on linear and exponential models with various optimization criterion. These models all focus on the problem of modeling the spectral counts in the mass spectrum of a protein. One important factor is clearly the amount of protein input to the mass spectrometer (since more input implies more output). However, other factors are important as well, such as the efficiency with which various peptides are produced, fractionated and ionized. Our goal is to determine how all these other factors are influenced by a peptide's amino-acid sequence. Once this is understood, it may be possible to predict the exact MS/MS spectrum of a protein. More importantly, it may also be possible to solve the inverse problem: given the MS/MS spectrum of a protein, estimate the amount of protein that was input to the mass spectrometer. This is a fundamental problem whose solution would enable biologists to determine the levels of the thousands of proteins in a tissue sample [47]. Besides the regression methods we developed, we also overview a classical technique developed in multivariate statistics called Canonical Correlation Analysis (CCA) that can be used to fit linear models to data in a more natural way. By proving an important technical result: theorem 1, we show that our linear model LIN3 is a more robust and efficient approximation of CCA.

As a first step in addressing these problems, we obtained a set of several thousand MS/MS spectra.[1] This data is the result of MS/MS experiments conducted on protein mixtures taken from various tissue samples of Mouse [47]. These high-throughput experiments provide a large amount of data on which to train and test data-mining methods. However, they also introduce a complication, since the amount of protein input to the mass spectrometer is unknown. Thus, it is in general unclear whether a high spectral count is due to the properties of the peptide or to a large amount of protein at the input.

[1] courtesy of the Emili Laboratory at the Banting and Best Department of Medical Research at the University of Toronto.

One of our challenges is to untangle these two influences. This distinguishes our work from other research in which the amount of protein is known *a priori* [21]. In effect, we must solve these two problems at once: the forward problem of estimating the spectrum of a protein, and the inverse problem of estimating the amount of protein given its spectrum. To deal with this complication, we treat the amount of protein as a latent, or hidden variable, whose value must be estimated. The data-mining methods presented in this work were developed and used, in part, because of the ease and simplicity with which this can be done. In addition, they lead to efficient algorithms based on well-developed operators of linear algebra (specifically, matrix inversion and eigenvector decomposition).

The thesis is organized as follows. Chapter 2 provides biological background. Chapter 3 outlines the discriminant analysis to the datasets. Chapter 4 introduces and evaluates our three models of MS/MS data. Chapter 5 outlines our methods for fitting Canonical Correlation Analysis models to data. Finally, Chapter 6 summarizes the results and suggests possible extensions for future work.

Chapter 2

Peptide Tandem Mass Spectrometry

2.1 Biological Background

[1]Tandem mass spectrometry involves several phases in which proteins are broken up and the pieces separated by mass [32, 46]. First, a complex mixture of thousands of unknown proteins is extracted from a cell culture or tissue sample. Since proteins themselves are too large to deal with, they are fragmented, producing a mixture of tens of thousands of unknown peptides. The peptides are then ionized and passed through a mass spectrometer. This produces a mass spectrum in which each spectral peak corresponds to a peptide. From this spectrum, individual peptides are selected for further analysis. Each such peptide is further fragmented and passed through a second mass spectrometer, to produce a so-called tandem mass spectrum. The result is a collection of tandem mass spectra, each corresponding to a peptide. Each tandem mass spectrum acts as a kind of fingerprint, identifying the peptide from which it came. By searching a database of proteins, it is possible to identify the protein that produced the peptide that produced the tandem mass spectrum. In this way, the proteins in the original tissue sample are identified. Often, the entire process is completely automatic.

[1]This chapter has been published on [5]

A peptide mixture is not analyzed all at once. Instead, to increase sensitivity, the peptides are "smeared out" over time (often using liquid chromatography), so that different kinds of peptides enter the mass spectrometer at different times. A typical MS/MS experiment may last many hours, with proteins and peptides being identified each second. Copies of a particular peptide may continue to enter the mass spectrometer for several seconds or minutes. As the copies enter, the peptide will be repeatedly identified, once a second. In this way, a peptide may be identified and re-identified many times, increasing the confidence that the identification is correct [20]. Each identification of a peptide is called a *spectral count*, since it requires the generation of a tandem mass spectrum. A large spectral count indicates that a peptide has been confidently identified [41, 35, 1].

However, *identifying* proteins is only part of the story. It is also important to *quantify* them, that is, to estimate how much of each protein is present in a cell. For this, we should develop mathematical models which can model the peak intensities accurately. The spectral count is influenced by the amount of protein in the input mixture. However, the exact mechanism determining the whole process is poorly understood [21]. In an ideal experiment, there would be no loss during digestion, fractionation and ionization. So, in the MS/MS spectrum for a given protein, one would expect that each peak would contain one peptide molecule for each protein molecule in the input. Consequently, an ideal spectrum for a given protein would consist of equal spectral counts. However, this is not observed experimentally. Figure 2.1 illustrates the differences between an ideal MS/MS spectrum and an experimental one.

Numerous reports in the literature address the question of what factors affect the quality of a MS/MS experiment [30] [43]. An obvious factor influencing spectral counts is the concentration of the peptides in the sample. However, it is not the sole factor. Other factors include sample preparation methods, the pH and composition of the solution containing the protein mixture [12] [43] [18] [8], the characteristics of the MS/MS apparatus [29] [38] [37], and the characteristics of the tissue sample being analyzed [21].

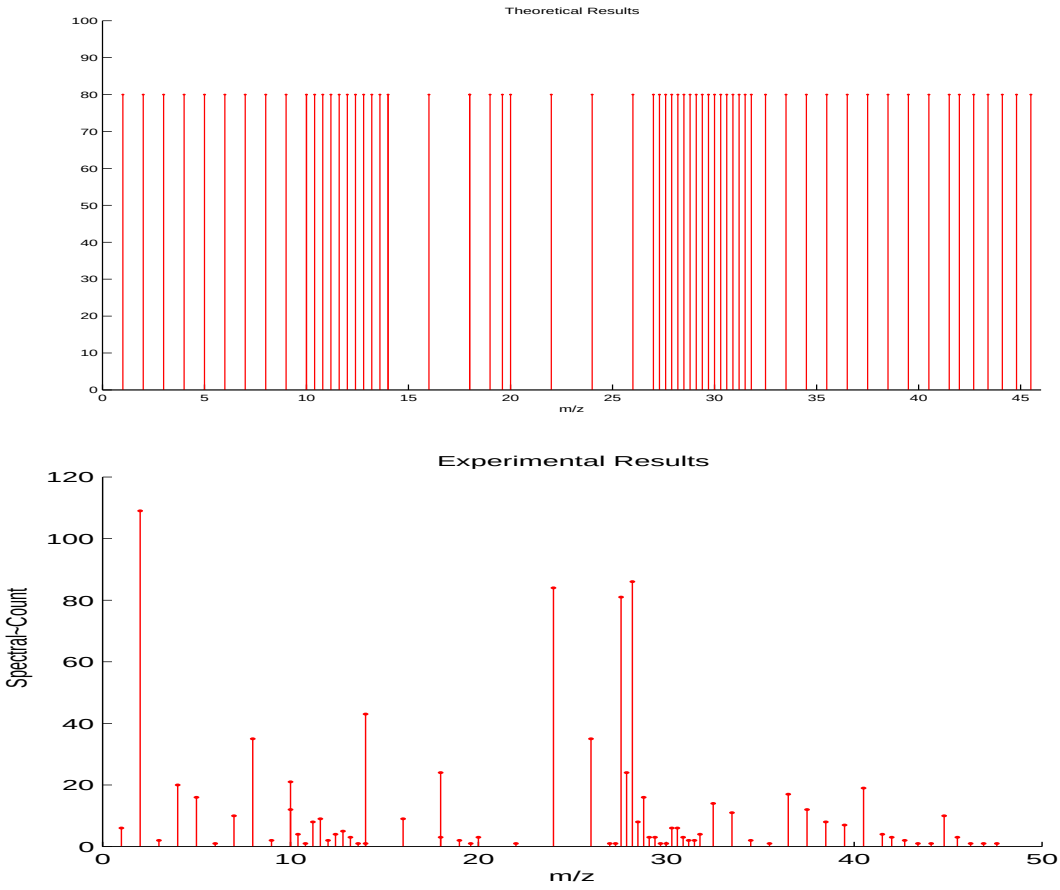

Figure 2.1: The upper panel is the theoretical MS/MS spectrum for the 47 peptides fragmented from protein MAP2-MOUSE digested with trypsin. The lower panel is an experimentally derived spectrum of the same protein.

These factors and others all affect the intensities of peaks in a MS/MS spectrum. Recent research has shown that the spectral counts of peptides are linearly related to protein abundance. In particular, as the relative abundance of a given protein is increased, the total spectral count of its peptides increases in direct proportion [33]. In effect, more input leads to proportionately more output. However, the relationship is not at all straightforward, since two proteins with the same spectral counts may have different abundances. Thus, despite the linear relationship, different proteins have different proportionality constants. Moreover, at present, there is no mathematical models to predict

what these constants are. That is, there is no complete quantitative theory relating a protein's abundance to its spectral count. Explaining why some peptides produce high spectral counts and some produce low spectral counts is a first step towards developing such a model, and is the goal of this work.

2.2 Real-World Datasets

To solve such kind of problems, the Emili Laboratory at the Banting and Best Department of Medical Research at the University of Toronto has provided us with several thousand MS/MS spectra. Table 2.1 shows a tiny sample of this data. (Details on how this data was generated can be found in [47].) The first column in the table is the Swissprot accession number identifying a protein. The second column is the amino-acid sequence of a peptide fragmented from the protein. The third column correlates well with the spectral counts produced by the peptide. (Its values are integers because it represents a count of peptide molecules.) The last column represents the charge of the peptide ion, which is typically 1, 2 or 3. Notice that there may be many entries for the same protein, since a single protein can produce many peptides.

Table 2.1: A fragment of the original data file

Protein ID	Peptide Sequence	Spectral Count	Charge
⋮	⋮	⋮	⋮
Q91VA7	$TAAARHCCNNLVIIR$	4	2
Q91VA7	$KLDCCCLFACAVHVK$	3	2
⋮	⋮	⋮	⋮

In fragmenting the proteins to produce peptides, the proteins were digested by the enzyme trypsin, which cuts c-terminal to lysine (K) or arginine (R). Trypsin may oc-

casionally cut at other locations, and other enzymes may nick the protein; hence, we sometimes see partial tryptic peptides (a K or R at the c-terminus, or flanking the peptide at the N-terminus).

The experimental results in this work are based on tables of real-world data similar to Table 2.1. They consist of three datasets derived from tissue samples taken from Mouse and were provided by the Emili Laboratory at the Banting and Best Department of Medical Research at the University of Toronto. We refer to these data sets as **Mouse Brain Data**, **Mouse Heart Data**, and **Mouse Kidney Data**. The Brain data set contains 10,786 peptides, with peak intensities ranging from 1 to 2,500; the Heart data set contains 9,623 peptides, with peak intensities from 1 to 1,996; and the Kidney data set contains 8,791 peptides, with peak intensities from 1 to 1,491.

2.3 Distribution of the MS/MS Data

Histograms of the spectral counts for the three data sets are shown in Figure 2.2. These show that the peak intensities are far from being uniformly or normally distributed. Instead, spectral counts are heavily concentrated near the minimum value of 1, and their frequency rapidly falls off as spectral count increases. For example, although the maximum spectral count in the Kidney dataset is 1,491, the median value is only 2! That is, half the peptides have spectral counts of only 1 or 2, while the remaining peptides have values with spectral counts from 2 to 1,491. Thus, the data values range over several orders of magnitude, with the bulk of the data concentrated at lower values. This distribution is likely due to a number of factors, including the distribution of protein levels in the tissue samples, and the distribution of ionization efficiencies among peptides. Untangling these factors is one of the goals of this work.

A more accurate description of the distribution of spectral counts is provided by the probability plots in Figure 2.3, which display values in sorted order. The two panels in

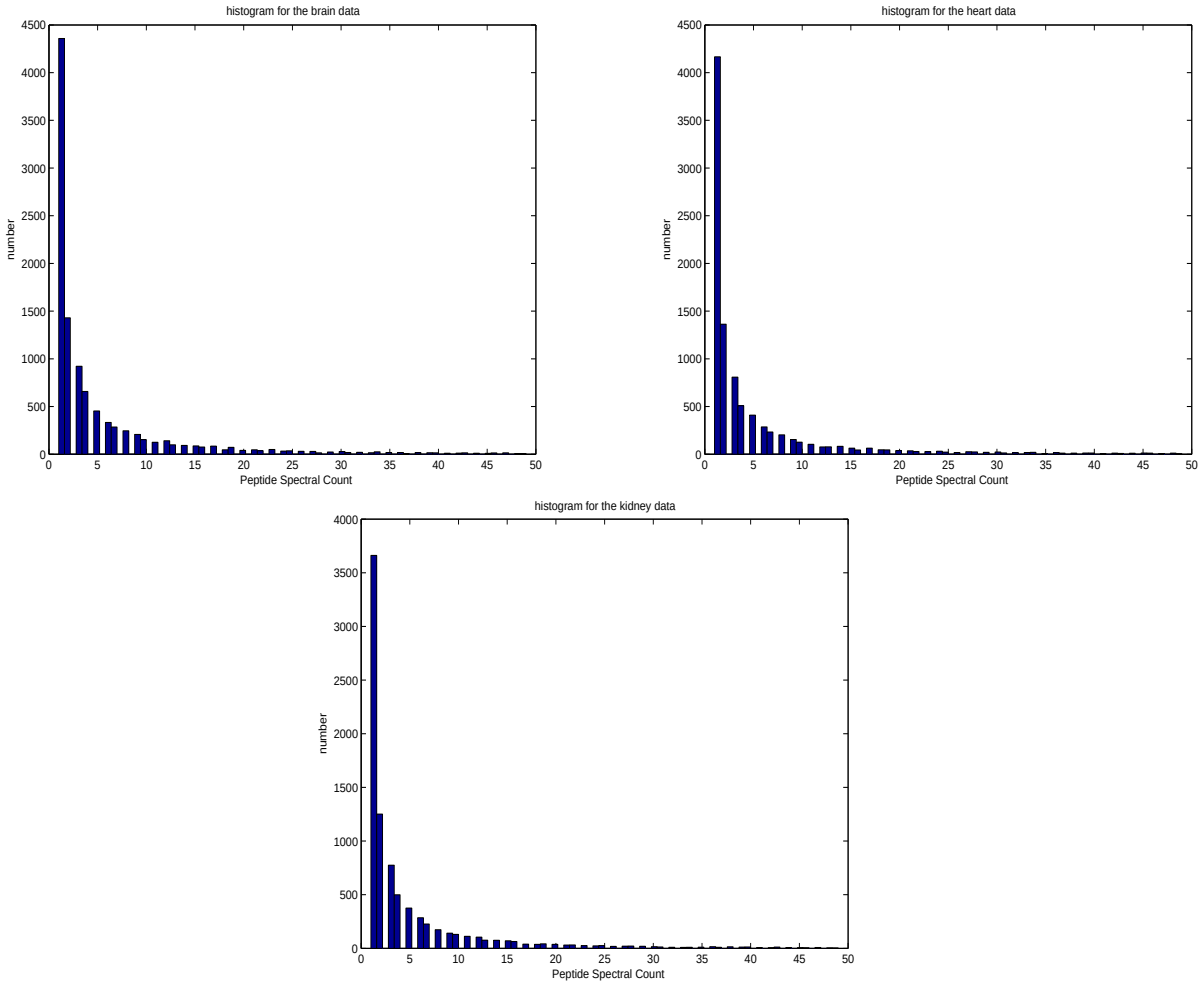

Figure 2.2: Histograms of peptide spectral counts. The upper left histogram is for the Brain dataset, the upper right is for Heart, and the lower one is for Kidney.

Figure 2.3 show probability plots for the reciprocal of spectral count; that is, they are probability plots of $1/y$, where y is spectral count. The vertical axis is the value of $1/y$, and the horizontal axis is the rank of this value. The plot in the left panel includes all observed spectral counts, while the plot in the right panel excludes lower level spectral counts, that is, with an intensity below 10. The horizontal line segments that make up the plots, especially at the upper end, are due to the discrete nature of the data (*i.e.*, y is an integer). However, the trend in the right hand plot is clearly a diagonal line, from the lower left corner of the plot to the upper right corner. This indicates that

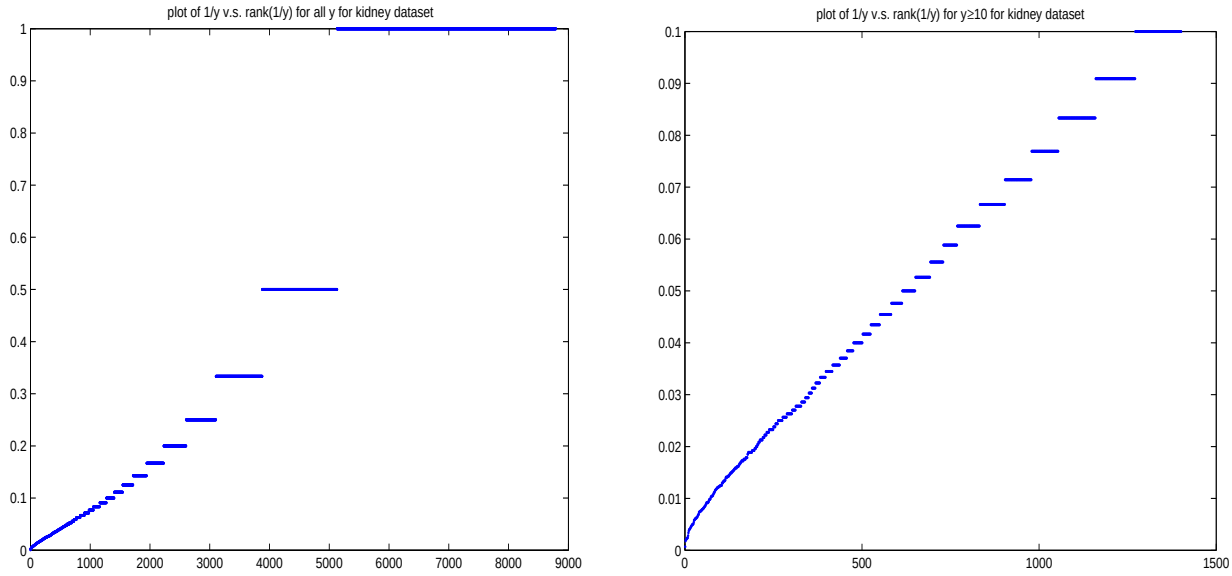

Figure 2.3: Probability plots of $1/y$ for the Kidney dataset, where y is spectral count. The left plot includes all observed values of y, while the right plot excludes values of y less than 10.

the cumulative distribution function of $1/y$ is a diagonal line, which means that $1/y$ is uniformly distributed, which in turn means that y has an $O(1/y^2)$ distribution [44]. The left hand plot is the same except that it curves up for large values of $1/y$, that is, for low peak intensities. Thus, while the probability density of $1/y$ is flat for $y \geq 10$, it decreases for $y < 10$. This in turn means that, while the distribution of peak intensities is $O(1/y^2)$ for $y \geq 10$, it is less than this for $y < 10$. Similar results hold for all three data sets, Kidney, Brain and Heart.

Chapter 3

Discriminant Analysis

3.1 Why Discriminant Analysis

[1]In this chapter, we focus on the problem of explaining why different peptides have different spectral counts. One important factor is clearly the amount of protein input to the mass spectrometer (since more input implies more output). However, other factors are important as well, such as the efficiency with which various peptides are produced, fractionated and ionized. Our goal is to determine how all these other factors are influenced by a peptide's amino-acid sequence. Once these influences are understood, it may be possible to predict the exact MS/MS spectrum of a protein, including the values of all its spectral counts. More importantly, it may also be possible to solve the inverse problem: given the MS/MS spectrum of a protein, estimate the amount of protein that was input to the mass spectrometer. This is a fundamental problem whose solution would enable biologists to determine the levels of the thousands of proteins in a tissue sample [7].

As a first step in explaining why different peptides produce spectral counts of different values, we reduce the problem to its simplest terms: explaining why some peptides produce peaks of higher spectral counts, while others produce lower spectral counts.

[1]This chapter has previously been published in [4]

Moreover, we treat this as a classification problem. That is, given the amino-acid sequence of a peptide, we attempt to classify the peptide as either high spectral counts or low spectral counts. If we can do this reliably, then we have effectively figured out what it is about the amino-acid sequence of a peptide that determines whether it produces high- or low-spectral counts. This is especially true if the parameters of the classifier can be interpreted biologically. To tackle this classification problem, this chapter evaluates and compares a variety of classification methods on MS/MS data. Conducting the evaluation involves three main steps: producing a set of labeled data, training a number of different classifiers on a portion of the data (the "training data"), and evaluating the effectiveness of the classifiers on the remaining data (the "testing data").

To produce a labeled set of data, we first obtained a set of several thousand MS/MS spectra.[2] From these spectra, we produced two classes of peptides, those with high spectral counts, and those with low spectral counts. One complication is that in the high-throughput MS/MS experiments from which the data was derived, the amount of protein input to the mass spectrometer is unknown. Thus, it is in general unclear whether a high spectral count is due to the properties of the peptide (which we are interested in) or simply due to a large amount of protein at the input (which we are not interested in). We resolved this problem by focusing on the spectra one protein at a time. In the spectrum of a single protein, all the spectral counts are derived from the same (unknown) amount of protein, so differences in spectral counts are not due to differences in protein input. In this way, by picking the highest and lowest peaks in each spectrum, we produced two classes of peptides, one class labeled "high spectral counts" and the other labeled "low spectral counts".

Since this is an initial study, we chose a number of basic classification methods which are extensively used in the data-mining literature: K-Nearest Neighbours (KNN), Lo-

[2]Courtesy of the Emili Laboratory at the Banting and Best Department of Medical Research at the University of Toronto

gistic Regression, Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Naive Bayes, and Hidden Markov Models (HMMs). Most of these methods have also been used in the context of microarray data to distinguish various cancer types [14]. To evaluate the methods for our application, we first trained them on two thirds of the labeled data, and then evaluated them on the remaining one third. To compare the methods, we produced both ROC curves and a table of classification error rates. In general, classification error rates depend on the size of the labeled classes, and can be deceptively low when the classes have very different sizes (as ours do, since most peptide peaks are low-intensity, not high-intensity). To compensate for this, we used the ROC data to estimate classification error rates for classes of equal size. In addition, the ROC curves themselves are insensitive to differences in class size. In this way, we can compare the performance of the classifiers on various data sets without having to worry about variations in class size. The ROC curves also allow us to conveniently compare classifiers for many different values of a discriminant threshold parameter.

Finally, it is worth noting that of the six classification methods we chose to evaluate, three are designed for classifying vectors (LDA, QDA and Logistic Regression), two are most-easily applied to vectors (KNN and Naive Bayes), while only the last (HMM) is specifically designed for classifying sequences. HMMs can therefore be applied directly to the amino-acid sequences of peptides (which is why we chose them). To apply the other five methods, we first converted the peptide sequences to feature vectors. Since there is a certain loss of information in this conversion, one might expect HMMs to perform the best, especially since they have achieved considerable success in other areas of sequence classification, such as speech recognition [42] and DNA sequence analysis [40]. Surprisingly, we found that for our peptide-classification problem, HMMs performed the worst. We discuss reasons for this in Section 3.4.

This chapter is organized as follows: Section 3.2 reviews the six classification methods we evaluate; Section 3.3 describes our data and our experimental design; Section 3.4

presents and discusses our experimental results; and Section 3.5 presents conclusions.

3.2 Discrimination Methods

Each spectral count in a mass spectrum is due to a particular peptide. The goal of this chapter is to explain why some peptides produce high spectral counts, while others produce low spectral counts. Specifically, we seek a method that, given the amino-acid sequence of a peptide, can classify the peptide as low spectral count or high spectral count. To this end, we trained and tested a variety of well-known classification methods on MS/MS data. Each of the methods can be used to assign objects to one of several classes, though we use them in a strictly binary mode, to assign peptides to one of two classes, which we refer to simply as Class 1 and Class 2, respectively. In our application, Class 1 is the set of peptides that produce high spectral counts, and Class 2 is the set of peptides that produce low spectral counts. How the peptides are represented depends on the classification method used. For one of the methods (Hidden Markov Models), each peptide is represented by its amino-acid sequence. For the remaining methods, each peptide is represented by a vector, $\mathbf{x}$, of features extracted from the sequence. How these features are extracted is described in Section 3.3. The rest of this section outlines all the methods used in our study.

Although the methods differ greatly, they all produce classifiers that are discriminative [36, 34, 45]. That is, given an object, the classifier produces a score or likelihood, p_1, that the object belongs to Class 1, as well as a score, p_2, that the object belongs to Class 2. A natural decision rule is to assign an object to Class 1 if $p_1 > p_2$, and to Class 2 otherwise. More generally, the error rates can be adjusted by introducing a decision threshold, t, so that an object is assigned to Class 1 if and only if $p_1/p_2 > t$. Of course, this decision rule is equivalent to $f(p_1/p_2) > t$, where f is any monotonically increasing function. Often, f is the logarithmic function, in which case the rule

becomes $l_1 - l_2 > t$, where $l_k = \log p_k$.

3.2.1 Naive Bayes

At an abstract level, most of the methods considered in this chapter use the same method to classify an object, $\mathbf{x}$. First, Bayes Rule is used to compute $p_k(\mathbf{x})$, the posterior probability that $\mathbf{x}$ belongs to class k:

$$p_k(\mathbf{x}) = \frac{\pi_k f_k(\mathbf{x})}{\sum_j \pi_j f_j(\mathbf{x})}$$

Here, π_k is the prior probability that $\mathbf{x}$ belongs to class k, and $f_k(\mathbf{x})$ is the conditional probability of $\mathbf{x}$ assuming class k. Typically, π_k is estimated as the proportion of data points in class k. Once the posterior probabilities are estimated, they are used in a likelihood ratio test to assign $\mathbf{x}$ to a class. In our application, for which $K = 2$, $\mathbf{x}$ is assigned to Class 1 if and only if $p_1(\mathbf{x})/p_2(\mathbf{x}) > t$, where t is a decision threshold.

The various classification methods differ primarily in their assumptions about the prior probabilities, $f_k(\mathbf{x})$. In the Naive Bayes classifier, it is assumed (simplistically) that the features of the vector $\mathbf{x}$ can be treated as independent random variables, so that f_k can be factored into a product of distributions, one for each component of $\mathbf{x}$. Thus, if $\mathbf{x} = (x_1, ..., x_n)$, then

$$f_k(\mathbf{x}) = \prod_i f_{ki}(x_i)$$

where $f_{ki}(x_i)$ is the (marginal) probability of x_i for class k. Despite this simplistic assumption, Naive Bayes classifiers often perform surprisingly well [24].

Originally, univariate Gaussians were used to estimate f_{ki}, though kernel density estimates are now common [24]. In our application, the components of the feature vector, $\mathbf{x}$, are discrete (as we shall see), and the most common values are 0 and 1, so we use binary distributions to estimate f_{ki}.

3.2.2 Linear Discriminant Analysis

In Linear Discriminant Analysis (LDA), each class is modelled by a multivariate Gaussian distribution, where each class is assumed to have the same covariance matrix. This assumption reduces the number of parameters that need to be estimated and also leads to a simple decision rule. In fitting the Gaussian distributions to the data, LDA produces maximum likelihood estimates for several parameters: π_k, the prior probability of class k; μ_k, the mean of class k; and Σ, the common covariance matrix. These estimates are given by the following formulas:

- $\hat{\pi}_k \;=\; N_k/N$
- $\hat{\mu}_k \;=\; \sum_i \mathbf{x}_{ik}/N_k$
- $\hat{\Sigma} \;=\; \sum_k \sum_i (\mathbf{x}_{ik} - \hat{\mu}_k)(\mathbf{x}_{ik} - \hat{\mu}_k)^T/(N-K)$

Here, K is the number of classes ($K = 2$ in our application), N_k is the number of feature vectors in the training data for class k, N is the total amount of training data (so $N = N_1 + \cdots + N_K$), and x_{ik} is the i^{th} feature vector in the training data for class k. If the feature space has dimension n, then LDA must estimate n parameters for each mean, and $O(n^2)$ parameters for the common covariance matrix, for a total of $O(Kn+n^2)$ parameters.

To classify a new vector, $\mathbf{x}$, LDA uses these parameter estimates and Bayes rule to compute $p_k(\mathbf{x})$, the posterior probability that $\mathbf{x}$ belongs to class k. It then performs a likelihood ratio test. In our application, for which $K = 2$, the test is $p_1(\mathbf{x})/p_2(\mathbf{x}) > t$, where t is a decision threshold. Because the class distributions are Gaussian and the covariance matrices are equal, it is not hard to show that

$$\log[p_1(\mathbf{x})/p_2(\mathbf{x})] \;=\; \mathbf{x} \bullet \mathbf{w} + b$$

where $\bullet$ denotes the dot product (or inner product) of two vectors, $\mathbf{w} \;=\; \hat{\Sigma}^{-1}(\hat{\mu_1} - \hat{\mu_2})$, and b is a constant, independent of $\mathbf{x}$ [24]. Thus, the likelihood ratio test reduces to a

particularly simple form: $\mathbf{x} \bullet \mathbf{w} > t$. The decision boundary between the two classes is therefore linear, and more specifically, it is a hyperplane normal to $\mathbf{w}$. It is interesting to note that changing the decision threshold, t, is equivalent to moving the hyperplane. Since $\mathbf{w}$ does not change, the orientation of the hyperplane remains constant while it moves in the direction of $\mathbf{w}$.

3.2.3 Quadratic Discriminant Analysis

Quadratic Discriminant Analysis (QDA) is a generalization of LDA. Like LDA, QDA models each class by a multivariate Gaussian distribution. However, it does not assume that each class has the same covariance matrix. In fitting the Gaussian distributions to the data, QDA produces the same maximum likelihood estimates for π_k, the prior probability of class k, and for μ_k, the mean of class k. However, it produces the following estimate for Σ_k, the covariance matrix of class k:

- $\hat{\Sigma_k} = \sum_i (\mathbf{x}_{ik} - \hat{\mu}_k)(\mathbf{x}_{ik} - \hat{\mu}_k)^T/(N_k - 1)$

In this case, it is not hard to show that

$$\log[p_1(\mathbf{x})/p_2(\mathbf{x})] = \mathbf{x}^T\mathbf{W}\mathbf{x} + \mathbf{x} \bullet \mathbf{w} + b$$

where $\mathbf{W} = (\hat{\Sigma}_2^{-1} - \hat{\Sigma}_1^{-1})/2$, $\mathbf{w} = \hat{\Sigma}_1^{-1}\hat{\mu}_1 - \hat{\Sigma}_2^{-1}\hat{\mu}_2$, and b is a constant, independent of $\mathbf{x}$ [13]. The likelihood ratio test thus becomes $\mathbf{x}^T\mathbf{W}\mathbf{x} + \mathbf{x} \bullet \mathbf{w} > t$, where t is a decision threshold. The decision boundary between the two classes is therefore quadratic, and can in general be a hypersphere, a hyperellipsoid, a hyperparaboloid, a hyperhyperboloid, or any combination thereof [13]. Note that if $\hat{\Sigma}_1 = \hat{\Sigma}_2$, then $\mathbf{W} = 0$, in which case the decision boundary is a hyperplane and the decision rule reduces to LDA.

If the feature space has dimension n, then QDA must estimate $O(n^2)$ parameters for each covariance matrix, for a total of $O(Kn^2)$ parameters, v.s. $O(n^2)$ for LDA. QDA thus admits a much wider range of decision surfaces than LDA, but at the cost of estimating K times as many parameters.

3.2.4 Logistic Regression

Like QDA, Logistic Regression can be viewed as a generalization of LDA. However, whereas QDA has more parameters than LDA, Logistic Regression has fewer. Like LDA, Logistic Regression provides a linear decision boundary between classes. The main difference is that Logistic Regression is in a sense more direct. Instead of first fitting multivariate Gaussians to each class (which requires estimating $O(n^2)$ parameters), Logistic Regression fits a linear decision directly to the data (which requires estimating only n parameters). More specifically, it assumes that for each feature vector $\mathbf{x}$, the log likelihood ratio is given by the equation $log[p_1(\mathbf{x})/p_2(\mathbf{x})] = \mathbf{x} \bullet \mathbf{w} + b$, for some vector $\mathbf{w}$ and some constant b. This equation is known to hold for a wide range of class density distributions. For instance, Section 3.2.2 showed that it holds for multivariate Gaussian distributions with equal covariance matrices. It is also known to hold for gamma distributions, exponential distributions, binomial distributions, Poisson distributions, and more generally, for any member of the general class of distributions known as the exponential family [26]. In this sense, Logistic Regression is more general than LDA.

It also requires estimating fewer parameters than LDA, as noted above. However, finding maximum likelihood estimates for the parameters of Logistic regression is more complex, since there are no closed-form formulas for them. Instead a set of nonlinear equations must be solved, using optimization techniques such as the Newton-Raphson algorithm [24].

3.2.5 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is different from the other methods considered in this paper in that it is *non-parametric*, so there are no parameters to estimate. Instead, the training data itself is used as a sample estimate of the underlying distributions of the two classes. The classification method is simple. Given an object, $\mathbf{x}$, to be classified, find the K objects

(or neighbors) in the training data that are closest to it. If most of these neighbors belong to Class 1, then $\mathbf{x}$ is assigned to Class 1; otherwise, it is assigned to Class 2. When $K = 1$, this method is called "Nearest Neighbor," or 1NN. In this case, $\mathbf{x}$ is assigned to the same class as the data point that is closest to it. One can view KNN as a voting system, in which some neighbors vote for Class 1 and others vote for Class 2. In its simplest incarnation, a simple majority vote is used, but this is not necessary. For example, one might require a super-majority vote in which $\mathbf{x}$ is assigned to Class 1 if and only if 2/3 of its neighbors are in Class 1. More generally, $\mathbf{x}$ can be assigned to Class 1 if $K_1/K_2 > t$, for some t. Here, K_1 is the number of K-nearest neighbors in Class 1, and K_2 is the number in Class 2 (so $K_1 + K_2 = K$). Note that this is a discriminative classifier, where K_1 is interpreted as the likelihood that $\mathbf{x}$ is in Class 1, K_2 as the likelihood it is in Class 2, and t is the decision threshold.

In using KNN, one must choose a value for K. If there is no reason to prefer any particular value *a priori*, then one can train classifiers using several values of K, and choose the value that minimizes the testing error. That is the approach taken in this paper. One must also choose a measure of "distance" (or similarity) between two objects. If the objects are vectors, then several choices naturally suggest themselves, including Euclidean distance, correlation coefficient, and the angle between the vectors. This paper uses Euclidean distance.

3.2.6 Hidden Markov Models

Hidden Markov Models are the only method used in this chapter specifically designed to deal with sequences. A Markov Model (MM) can be thought of as a finite automaton with a probabilistic interpretation.[3] That is, it consists of a set of states and transitions between them; however, unlike ordinary automata, each transition in a MM has a probability associated with it. The result is that whereas an ordinary automata defines a *set*

[3]Strictly speaking, these are so-called first-order Markov Models.

of sequences, a Markov Model defines a *distribution* of sequences, since each sequence has a certain probability of being generated. A Hidden Markov Model (HMM) differs from a Markov Model in that in addition to transitions, each state also has a set of outputs symbols, each with an associated probability. When an HMM is in a given state, it chooses an output symbol probabilistically, and prints it. Whereas a MM generates a sequence of states, a HMM generates a sequence of output symbols. Like MMs, HMMs define a probability distribution over sequences, but since HMMs can generate sequences that MMs cannot, they are strictly more powerful.

Efficient algorithms have been developed for HMMs. For instance, given a sequence, $\mathbf{s}$, and a HMM, inference algorithms compute the probability, $\pi(\mathbf{s})$, that $\mathbf{s}$ will be generated by the HMM. Learning algorithms also exist for HMMs. Given a set of sequences, these algorithms will find the HMM with a given number of states that is most likely to generate the set. These algorithms are also used in sequence classification. First, a learning algorithm is used to estimate an HMM for each class. Next, to classify a particular sequence, $\mathbf{s}$, an inference algorithm is used to compute the prior probability, $\pi_k(\mathbf{s})$, that $\mathbf{s}$ belong to class k. Then, Bayes Rule is used to compute the posterior probability, $p_k(\mathbf{s})$, that $\mathbf{s}$ belongs to class k. Finally, a likelihood ratio test is used to assign $\mathbf{s}$ to a class.

3.3 Study Design

As described in Section 3.1, we divide the peptides in the training data into two classes, those that produce high spectral counts, and those that produce low spectral counts. One complication is that in the high-throughput MS/MS experiments from which the data was derived, the amount of protein input to the mass spectrometer is unknown. Thus, it is in general unclear whether a high spectral count is due to the properties of the peptide (which we are interested in) or simply due to a large amount of protein at

the input (which we are not interested in). We resolved this problem by focusing on the spectra one protein at a time. In the spectrum of a single protein, all the spectral counts are derived from the same (unknown) amount of protein, so differences in peak intensity are not due to differences in protein input. In this way, by picking the highest and lowest spectral counts in each spectrum, we can factor out the effect of protein input levels.

To do this, we first identified those proteins that have at least two peptides with observable spectral counts in their spectra. We then built an index for these proteins, and a separate index for their peptides. For the Brain dataset, the indexes contain 8,527 peptides and 1,664 proteins, respectively. For the Heart dataset, the indexes contain 7,660 peptides and 1,281 proteins, respectively. For the Kidney dataset, the indexes contain 7,074 peptides and 1,291 proteins, respectively.

For each protein, we then picked the peptides with the highest and lowest spectral peaks. Moreover, we did this only for proteins with a wide range of spectral counts, so that we could be sure of obtaining peptides with genuinely low- and high-spectral counts. We did this in two different ways, producing two pairs of peptide classes, which we call MAX-MIN and STD-DEV, respectively. The two pairs differ only in the proteins selected. For MAX-MIN, a protein is selected if (i) the highest spectral count in its spectrum is greater than 12, and (ii) the value of this highest spectral is at least 12 times greater than the count of the lowest peak. For the STD-DEV, a protein is selected if the standard deviation of all its peak intensities is greater than 4. Both MAX-MIN and STD-DEV contain two classes of peptides, one with high spectral counts, and one with low spectral counts. In MAX-MIN, the two classes of peptides are guaranteed to have very different spectral counts, since the high spectral counts are guaranteed to be at least 15 times greater than the low spectral counts. In STD-DEV, the peptides in the two classes do not always have such a great difference in spectral counts, but they contain more peptides, *i.e.*, more data on which to train and test the classifiers. In this way, from the original, unlabeled Kidney dataset, we generated two labeled datasets, one

using MAX-MIN and one using STD-DEV. Likewise for the Heart and Brain datasets, for a total of six labeled datasets.

The sizes of the datasets generated by the MAX-MIN method are as follows: the Brain dataset has 658 peptides in the "high spectral counts" class, and 1567 in the "low spectral counts " class; the Heart dataset has 389 peptides in the "high spectral counts" class, and 1232 in the "low spectral counts " class; the Kidney dataset has 470 peptides in the "high spectral counts" class, and 1,211 in the "low spectral counts " class. The sizes of the datasets generated by the STD-DEV method are as follows: the Brain dataset has 818 peptides in the "high spectral counts" class, and 1,735 in the "low spectral counts " class; the Heart dataset 463 peptides in the "high spectral counts" class, and 1,297 in the "low spectral counts " class; the Kidney dataset has 550 peptides in the "high spectral counts" class, and 1,327 in the "low spectral counts " class;

We estimated the generalization error of the methods in two ways. In the first approach, we divided each of the six labeled datasets randomly into training and testing data, in a 2:1 ratio. We then trained and tested each classification method on each of the six datasets. The results show the variation in performance of the methods over different datasets and different data-generation methods. In the second approach, we used only one of the six datasets: the Kidney dataset generated by STD-DEV. On this dataset, we did 10-fold cross validation for each classification method. The results show the variation in performance of each method over different splits in the dataset.

Finally, for many of the classification methods used in this study, we must represent each peptide as a vector, $\mathbf{x}$. We have two ways of doing this, using vectors with 21 and 421 components, respectively. The 21-component vectors represent the amino-acid composition of a peptide. Since there are twenty different amino acids, the vector has 20 components, $(x_1, ..., x_{20})$, where the value of x_i is the number of occurrences a particular amino acid in the peptide. In addition, the vector has a 21^{st} , x_0, whose value is always 1, to represent a bias term, as is common in machine learning models [24]. The 421-

component vector includes the original 21 components plus 400 more representing the dimer composition of a peptide. A *dimer* is a sequence of two amino acids, and since there are 20 distinct amino acids, there are 400 distinct dimers. This, larger vector representation was used only with the Naive bayes classifier, because of its reputation for working well in high dimesnions. The other vector-based classifiers were used only with 21-component vectors, to prevent overfitting. For comparison purposes, Naive Bayes was sometinmes also used with this shorter vector representation.

3.4 Results and Analysis

Figures 3.1 through 3.4 show ROC curves evaluating the performance of each of the classification methods under various conditions. In an ROC curve, the horizontal axis is referred to as specificity, and the vertical axis as sensitivity [24]. In our case, specificity is the probability that a peptide with spectral counts is predicted to have high spectral counts. Likewise, sensitivity is the probability that a peptide with low spectral counts is predicted to have low spectral counts. Each ROC curve corresponds to a single classification method, and each point in an ROC curve corresponds to the method being used with a different decision threshold.

Figure 3.1 shows ROC curves for K-Nearest Neighbors and Hidden Markov Models, with different values for K and number of states, respectively. The six subfigures represent experimental results for each of the two methods on each of the three datasets generated by the STD-DEV method. Note that in these figures, K-NN performs best when K=50, except on the heart dataset, where 10-NN performs better. Likewise, HMM performs best when it has only 1 state, except on the brain dataset, where it performs best with 5 states. Figure 3.2 shows ROC curves for all six classification methods on all six labeled datasets. In the top half of the figure, the datasets were generated by the MAX-MIN method. In the bottom half, they were generated by the STD-DEV method.

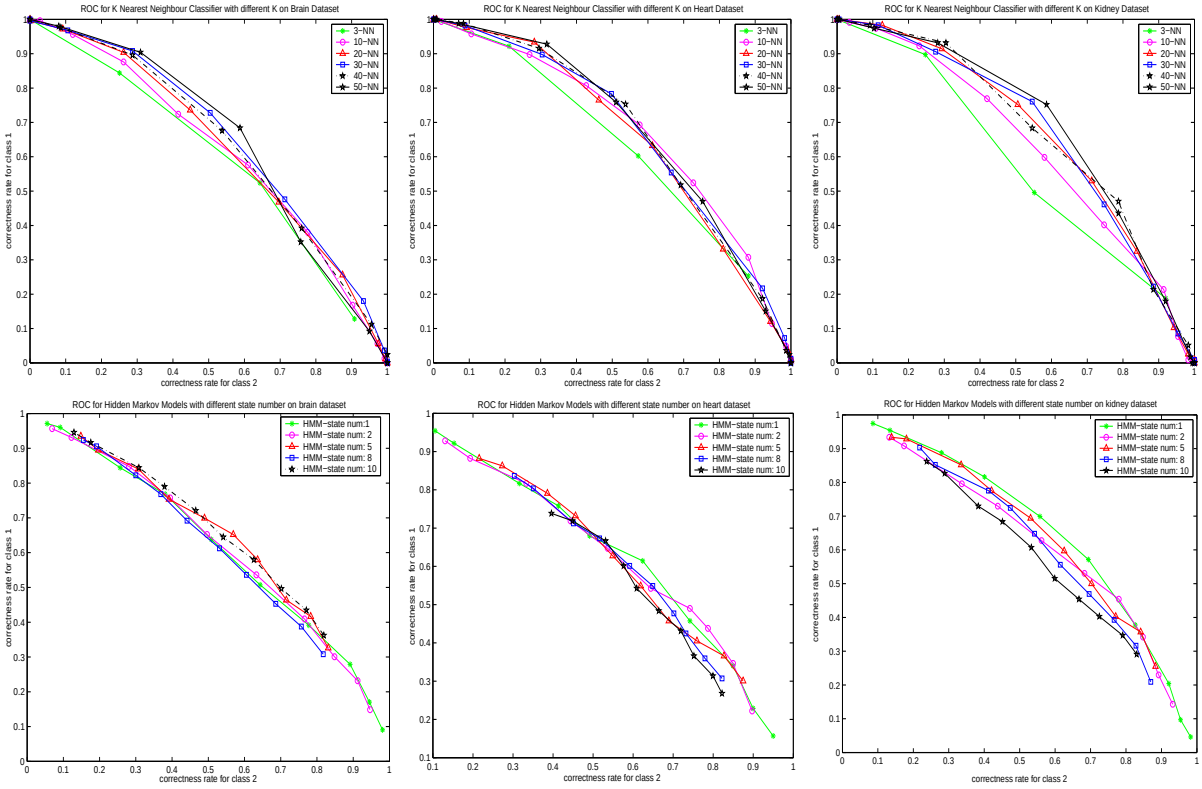

Figure 3.1: ROC curves illustrating the performance on three different datasets of the KNN and HMM classifiers. The top panels show KNN with six different values of K. The bottom panels show HMM with five different numbers of states. The left figures are for the Mouse brain dataset, the middle figures are for the Mouse heart dataset, and the right figures are for the Mouse kidney dataset. All three datasets were generated by the STD-DEV method.

Except for Naive Bayes, which used the 421-component vectors, all the vector-based classifiers in this figure used 21-component vectors.

Figure 3.3 shows the results of cross validation on each of the six methods. Here, KNN is used with K=50, and HMM is used with a single state, the values for which they performed best in the earlier experiments. Each of the six subfigures contains ten ROC curves, one for each run of 10-fold cross validation. The one exception is the subfigure for the Naive Bayes classifier, which contains twenty ROC curves, ten for Naive Bayes used

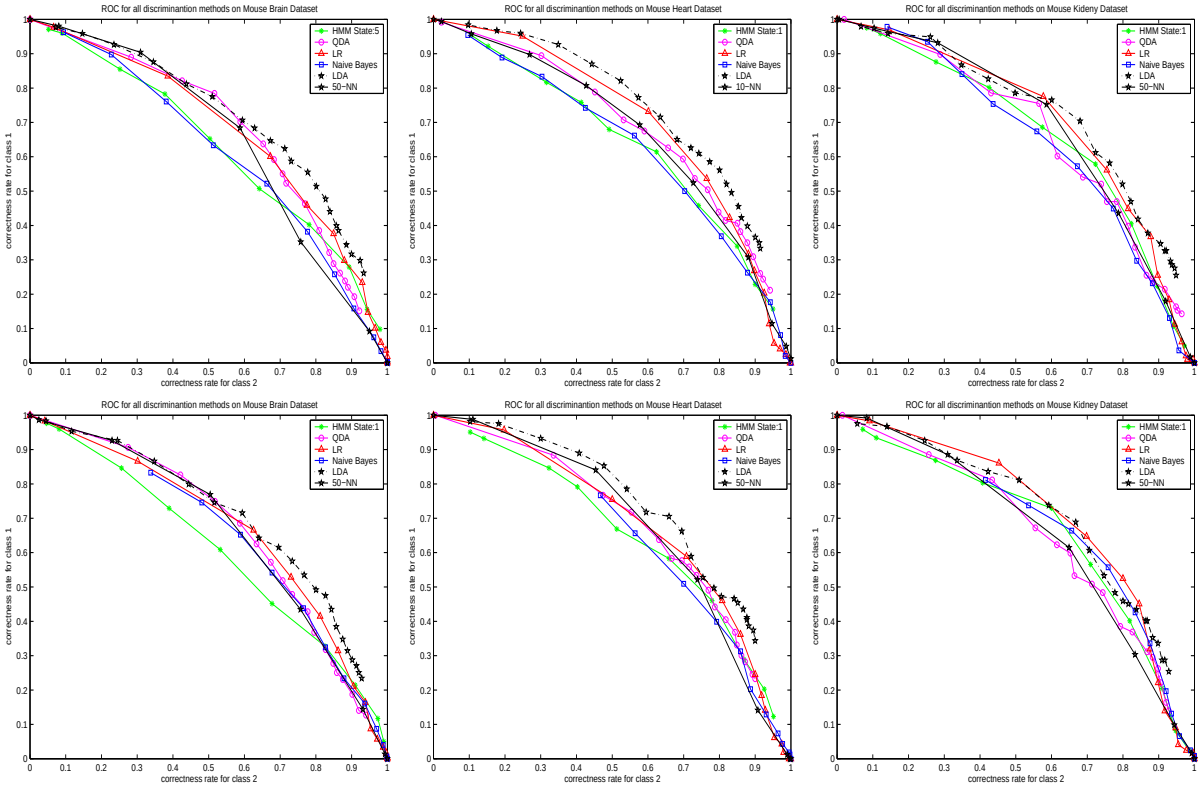

Figure 3.2: Comparison of all the discrimination methods on three datasets. The top panels are for datasets generated by the MAX-MIN method, and the bottom panels are for datasets generated by the STD-DEV method.

with 21-component vectors, and ten for 421-component vectors. The other vector-based classifiers were all used with 21-component vectors. The curves in Figure 3.3 illustrate the variability in our estimates of generalization error, variability due to the splitting of the data into training and test sets. Figure 3.4 shows average ROC curves for each classification method. Each point in an average curve is the mean of ten corresponding points in ten curves in Figure 3.3. These average curves provide an estimate of the average generalization error of each classification method.

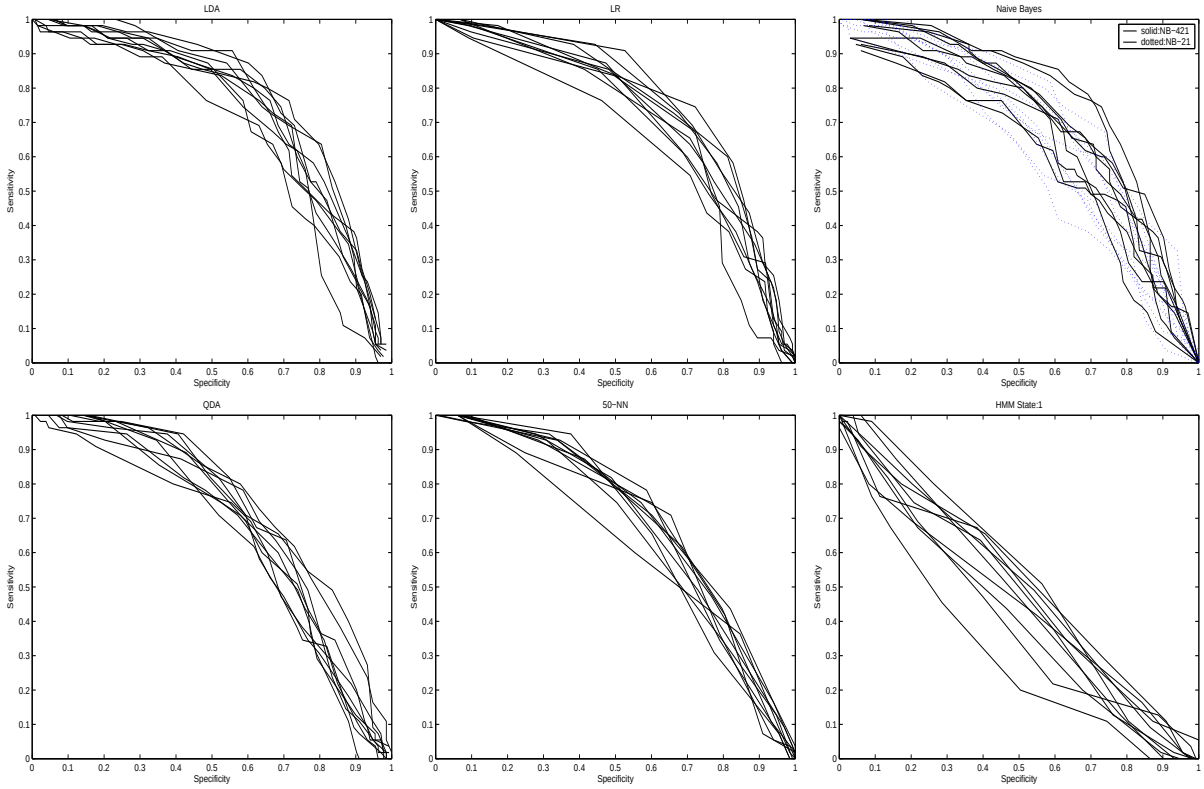

Figure 3.3: Results of the cross-validation experiments on all the discrimination methods.

3.4.1 ROC Curves and Evaluation

From the ROC curves, several conclusions are immediately apparent: (*i*) LDA performs the best, (*ii*) Hidden Markov Models perform the worst, (*iii*) the variability in the other methods is large compared to the differences in their average performance, and (*iv*) except for HMM, all the methods performed better with a 9:1 split of training to testing data than with a 2:1 split. This last observation is probably due to the larger portion of training data available in the 9:1 split.

The simple LDA method produced impressively better performance than other, more-sophisticated methods, such as QDA and especially Hidden Markov Models. With the exception of the Mouse Kidney dataset generated by STD-DEV, on which Logistic Regression performed best, LDA performed better than all other methods on all datasets. Many of the other methods appear to be overfitting, which may be why their perfor-

Table 3.1: Test set Error. Of the Misclassified Samples for 6 Discrimination Methods Applied to 4 Datasets. Two Measurements for selecting high and low classes are used.

	Mouse Brain		Mouse Heart		Mouse Kidney	
	MAX-MIN	STD-DEV	MAX-MIN	STD-DEV	MAX-MIN	STD-DEV
LDA	0.3316	0.3446	0.3192	0.3178	0.3080	0.3218
QDA	0.3498	0.3635	0.3542	0.3632	0.3398	0.3744
LR	0.3632	0.3544	0.3336	0.3517	0.3237	0.3272
KNN	0.3641	0.3633	0.3651	0.3528	0.4056	0.3681
NB	0.4078	0.3794	0.3880	0.3832	0.3775	0.3397
HMM	0.4065	0.4235	0.3808	0.3791	0.3492	0.3352

mance on the testing data is worse than LDA. For example, since QDA has almost twice as many parameters as LDA, it may be overfitting. Likewise, Logistic Regression is more general than LDA and can require about 30% more data to obtain the same fit [24].

Although it is hard to rank methods other than LDA and HMM, it should be noted that Naive Bayes performed better with 421-component vectors than with 21-component vectors. This is apparent both in Figure 3.4 and in Figure 3.3. Of course, more parameters can always lead to a better fit on the training data, and with only 1000 to 1500 training points, Naive Bayes is in danger of overfitting when used with 421-component vectors. So, it is surprising that it performs well on the testing data, living up to its reputation of performing well in high-dimensional spaces. It may be possible to further improve the performance of Naive Bayes by using more-complex prior distributions for the individual features. At present, we use binary distributions. Since each feature represents an amino-acid count, a binary distribution effectively means that each amino acid is modeled as being either present or absent in a peptide. This represents a loss of information, which more-complex prior distributions could eliminate.

To our surprise, Hidden Markov Models performed the worst of all the classification

methods: sometimes even worse than randomly guessing. This is despite the fact that they work directly on the peptide sequences, and not on vectors with lower information content. One reason seems obvious from the ROC curves in Figure 3.1. These show that performance generally degrades as the number of states in the HMM increases. This is a sure sign of overfitting. In fact, a HMM with 10 states will have $10 + 10^2 + 10 \times 20 = 310$ parameters. The factor of 20 comes from the 20 amino acids that each of the 10 states must be able to output. Since the classes in our training data often have only about 300 samples, over fitting is surely taking place. Even with only 5 states, the number of parameters is $5 + 5^2 + 5 \times 20 = 130$, which is still too many. In addition to this, HMMs only capture the *proportion* of amino acids in a peptide sequence. In particular, with only a small number of states, they cannot capture the absolute number of amino-acid occurrences, as our vectors do. It seems reasonable to suppose that the actual presence of a particular number of certain acids may be an important factor in the ionization of peptides. In fact, in a separate experiment, we used vectors whose components represented amino-acid proportions, instead of absolute numbers, and this caused the performance of Logistic Regression to decrease greatly, to about the same level as HMMs.

3.4.2 Classification Error Rate

In general, classification error rates depend on the size of the labeled classes, and can be deceptively low when the classes have very different sizes (as ours do, since most peptide peaks are low-intensity, not high-intensity). Even random guessing can appear to produce very low error rates. To compensate for this, we used the ROC data to estimate classification error rates for classes of equal size. This is easily done with the following formula

$$\text{error rate} \quad = \quad 1 \; - \; \frac{\text{specificity} + \text{sensitivity}}{2} \tag{3.1}$$

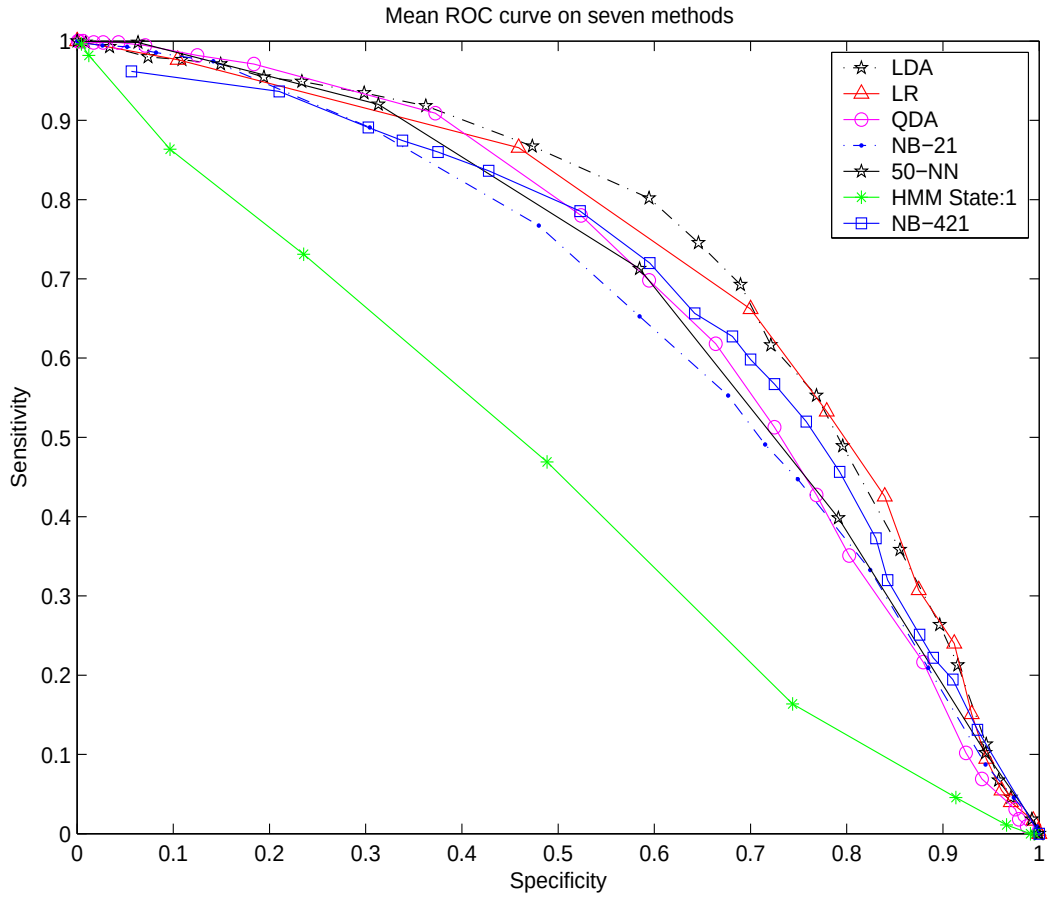

Figure 3.4: Mean ROC curves for the seven methods

We applied this formula to every point on every ROC curve in Figure 3.2. For each curve, we chose the lowest resulting error rate, to indicate the best possible performance for the classifier on the dataset. Table 3.4.1 shows the resulting classification error rates for each of the six classifiers on each of the three different datasets, with both MAX-MIN and STD-DEV used to choose proteins. The table shows that the classification error rates range from 0.3 to 0.4. In addition, it corroborates some of the results discussed above. For instance, LDA performs the best across all datasets, while HMMs perform the worst.

3.5 Conclusions of Discriminant Analysis

In this chapter, we applied several well-known classification methods to Peptide Tandem Mass Spectrometry data. We evaluated and compared the methods based on their ability to predict the level of spectral counts in a mass spectrum based on the amino-acid sequence of peptides. Specifically, the methods were required to divide peptides into two classes: those that produce high spectral counts, and those that produce low spectral counts.

Overall, we found that for our datasets, simple classifiers such as LDA performed well compared with more sophisticated ones, such as QDA and Hidden Markov Models. In the main comparison based on ROC evaluation, LDA has the best generalization performance. In decreasing order of performance, the other classifiers were Logistic regression, Naive Bayes (with 421-component vectors), QDA, KNN (with K=50), Naive Bayes (with 21-component vectors), and finally HMMs (with any number of states), which sometimes performed worse than random guessing. The performance of all the methods was quite variable. Misclassification rates for classes of equal size were also estimated, and they corroborate our conclusions drawn from the ROC evaluation.

Several further steps suggest themselves. For instance, the performance of each of the methods might be improved by boosting. Decision trees (which often work well with boosting) could be tried. Clustering methods could also be used to search for structure within each of the labeled classes, in order to provide better priors for the classification methods. In addition, Bayesian inference, bootstrapping and bagging could be used to deal with the problems of overfitting. To prevent over fitting of Hidden Markov Models, we are considering more-structured versions of them, such as Profile Hidden Markov Models, which have significantly fewer parameters. Especially the mixed Hidden Markov Models and some Bayesian analysis with some biological *a priori* may also be interesting. Finally, we are considering methods for extracting more training data from the MS/MS spectra. In this chapter, we considered two methods, MAX-MIN and STD-DEV, but

many other methods are also possible.

Chapter 4

Regression Analysis

4.1 Modeling the MS/MS Data

[1] In the high-throughput experiments we are dealing with, the amount of protein input to the mass spectrometer is unknown, our goal in this chapter is to develop methods to estimate it. Here, our working hypothesis is that for a given MS/MS experiment, each spectral count in a spectrum is produced independently of the others. That is, we assume that the value of the spectral count depends on only two factors: (*i*) the peptide responsible for this spectral count, and (*ii*) the (unknown) amount of protein input to the mass spectrometer. Although interactions between peptides do sometimes occur in mass spectrometer experiments, we use independence as a simplifying approximation in this initial study. More specifically, we assume that each peptide has an associated *ionization efficiency* that accounts for the relative values of the various spectral counts in a spectrum. The ionization efficiency represents the probability that a given peptide will enter the mass spectrometer as an ion. Since only these peptides contribute to the value of a spectral count, we expect that spectral count will be proportional to ionization efficiency. The ionization efficiency can be thought of as the propensity of a peptide to

[1] This chapter has appeared in [6]

ionize, though it can include other factors as well, such as the propensity of the peptide to be produced in the first place, *i.e.*, to be cleaved from the original protein. This idea is expressed formally by the equation $output = input \times ie$, where *output* is the value of a spectral count (measured in number of peptide molecules), *input* is the amount of protein from which the peptide was derived (measured in number of protein molecules), and *ie* is the ionization efficiency, a number between 0 and 1.

Under ideal conditions, each protein molecule would fragment into a set of peptide molecules, each of which would then ionize, enter the mass spectrometer, and contribute to a spectral count [48]. In this case, the ionization efficiency of each peptide would be 1, and the spectral counts for a given protein would all have the same values (equal to the amount of protein input). In practice, though, peptides do not all ionize with equal efficiency, and they produce spectral counts of different values [10]. In fact, some peptides do not ionize at all, so their ionization efficiency is 0 and they produce spectral count of value 0 [22]. The problem is to account for these differences and, more quantitatively, to estimate the ionization efficiency of each peptide. In data-mining terms, we want to learn a function, f, from peptides to the interval $[0, 1]$, where $f(p)$ is the ionization efficiency of peptide p. We shall identify peptides by their amino-acid sequence, *seq*, so $f(p)$ is really $f(seq)$.[2] The equation for peak intensity therefore becomes $output = input \times f(seq)$, where *output* and *seq* are observed, *input* is unobserved, and f is to be learned.

Note that once f is learned, we can apply it to new peptides to estimate the amount of protein input to the mass spectrometer, via the equation $input = output/f(seq)$. Our experiments in Sections 4.3 and 4.4 illustrate this idea by learning f on a training dataset and then applying it to a test dataset to estimate protein levels. Learning f is thus the central problem in estimating amounts of protein. This problem would be

[2]In fact, we are interested not just in peptides, but in peptide *ions*, which are identified by their sequence and their charge. However, to simplify the presentation, we leave it as understood that charge is included. Charge is explicitly included in our experiments, as described in Section 4.3.2.

relatively straightforward if the amounts of protein in the training dataset were known. In this case, we could estimate values of f for numerous values of seq, via the equation $f(seq) = output/input$, so learning f would be a regression problem. Unfortunately, the training data does *not* include information about protein amounts. In effect, the amounts of protein is a set of thousands of hidden variables, one for each protein. Dealing with these hidden variables is the main data-mining challenge addressed in this chapter.

4.1.1 Modelling Ionization Efficiency

Varying the form of the function f leads to a variety of different models of the data and to different learning problems. In this initial study, we consider three kinds of function:

$$\text{Linear}: f(seq) = \mathbf{x}\bullet\beta \qquad \text{Exponential}: f(seq) = e^{\mathbf{x}\bullet\beta} \qquad \text{Inverse}: f(seq) = 1/(\mathbf{x}\bullet\beta)$$

Here, $\mathbf{x}$ is a vector of peptide properties (derived from seq), β is a vector of parameters (to be learned), and $\bullet$ represents the dot product (or inner product) of the two vectors. The theoretical results in Section 4.2 show that for these three kinds of function, the learning problem reduces to tractable problems in linear algebra. The study design in Section 4.3 describes how the property vector, $\mathbf{x}$, is derived from a peptide sequence. Finally, the experimental results in Section 4.4 measure the biological realism of these functions and vectors.

Briefly, we investigate linear models because of their simplicity and because they are directly amenable to the techniques of linear algebra. We investigate exponential models because, by taking logs, they become linear. In addition, exponential models have the advantage that the ionization efficiency is guaranteed to be positive. In contrast (as we shall see in Section 4.4), the linear model may produce a preponderance of positive values, but it sometimes produces negative values as well, which are meaningless (though very small negative values can be assumed to be zero). As we shall see, both models

allow for efficient estimation of the (unknown) amount of protein input, though by very different means.

The inverse model has a different motivation. As we show in Chapter 2, spectral counts have a very skewed distribution of values, ranging over several orders of magnitude, with most of the values concentrated at the very low end of the spectrum. In fact, we show that the distribution is $O(1/y^2)$, where y denotes spectral count. It is very difficult to fit a linear model to data with this kind of distribution, since a small number of very large values tends to dominate the fit. Even if the largest values are removed, the next largest values dominate, ad infinitum. Taking logarithms helps, but even $\log(y)$ has a skewed distribution. However, $1/y$ has a uniform distribution, thus eliminating all skew. This is the motivation for the inverse model: to transform the data to a form that is more manageable. In addition, as we shall see in Section 4.2, all the methods we develop for fitting the linear model can easily be adapted to fit the inverse model.

4.1.2 Instantiating the Model

To keep track of different proteins and peptides, we use two sets of indices, usually i for proteins and j for peptides. Proteins are numbered from 1 to N, and the peptides for the i^{th} protein are numbered from 1 to n_i. Thus, seq_{ij} denotes the amino acid sequence of peptide number j of protein number i. Likewise, $\mathbf{x}_{ij}$ denotes the vector of properties for peptide j of protein i. In the sequel, we shall use y to denote the value of a spectral count. Thus, y_{ij} is the spectral count value of peptide j of protein i. We shall also use in to denote the (input) amount of a protein. Thus, in_i is the amount of protein i. With this notation, the equation for spectral counts described above becomes a set of equations:

$$y_{ij} \;=\; in_i \times f(seq_{ij}) \qquad\qquad for\ i\ from\ 1\ to\ N,\ \ and\ j\ from\ 1\ to\ n_i. \tag{4.1}$$

When instantiated with linear, exponential and inverse functions, they become, respectively,

$$y_{ij} = in_i \times (\mathbf{x}_{ij} \bullet \beta) \qquad y_{ij} = in_i \times e^{\mathbf{x}_{ij} \bullet \beta} \qquad y_{ij} = in_i / (\mathbf{x}_{ij} \bullet \beta)$$

Not all these equations contain useful information. In fact, only those proteins that produce at least two peptides can be used for estimating f. If a protein produces only one peptide, then $j = 1$ and the protein yields only one equation: $y_{i1} = in_i \times f(seq_{i1})$. This is the only equation containing the unknown value in_i. Once we know f, we can use this equation to estimate in_i. However, the equation provides absolutely no help in estimating f itself, since it does not constrain f in any way. In fact, it is trivially satisfied for *any* f by using $in_i = y_{i1}/f(seq_{i1})$. If all the proteins produced only one peptide, then we would have no way to estimate f, since any f would do. For this reason, our data-mining methods ignore all those proteins that produce only one peptide. In fact, the first method considered below explicitly requires at least two peptides per protein.

Finally, we note a fundamental limit to what can be learned from the data available to us. In particular, we can only learn the *relative* amount of a protein in a tissue sample, not the *absolute* amount. For instance, we could infer that the amount of protein 1 is twice that of protein 2, but we cannot infer exactly how much there is of either protein. This follows immediately from equation 4.1. Since in_i and f are both unknown, we can always scale one up as long as we scale the other down by the same amount. Specifically, suppose that $in_1, \cdots, in_N$ and f is a solution to the equation. Then $in'_1, \cdots, in'_N$ and f' is an equally good solution, where $in'_i = in_i \times c$ and $f'(seq) = f(seq)/c$, for all i and seq, and any constant c. Thus, there are infinitely many solutions, each predicting different absolute amounts for the proteins and different ionization efficiencies for the peptides. However, all these solutions predict the same relative values, since $in_i/in_j = in'_i/in'_j$ for any two proteins i and j, and $f(seq_i)/f(seq_j) = f'(seq_i)/f'(seq_j)$ for any two peptides i and j. In this way, the relative amounts of protein and the relative ionization efficiencies of peptides can be learned. Moreover, by using a small amount of calibration data,

we can convert these relative values into absolute ones. That is, if we are given the absolute amount of just a few proteins, we can then use the ratios in_i/in_j to estimate absolute values for all the proteins, from which we can estimate absolute values for all the ionization efficiencies.

4.2 Fitting the Models to Data

In the previous section, we developed three generative models of MS/MS data, linear, exponential and inverse. In this section, we develop methods for fitting these models to data, including methods for estimating the amount of protein input to the mass spectrometer.

An important parameter in any model of MS/MS data is the amount of each protein, in_i, input to the mass spectrometer. If this amount were known, then fitting our models to the data would be a straightforward problem of regression. If the fit is good, this would solve the biological problem of predicting the intensity of each spectral peak given the amino-acid sequence of the peptide and the amount of protein at the input. Unfortunately, the amounts of each protein are unknown. Moreover, there are thousands of proteins, each with a different input level, and the various input levels differ by several orders of magnitude. To deal with these complications, we treat the input levels, in_i, as latent, or hidden variables, whose values must be estimated. We therefore have thousands of hidden values to estimate, in addition to the parameters of our models. An accurate estimation of these values would solve another important biological problem: estimating the amounts of each of the thousands of proteins in a tissue sample given their MS/MS spectra. Moreover, since the protein levels differ so dramatically, even an approximate estimate would be biologically useful.

4.2.1 Linear Models

In this section, we develop a family of methods for fitting the linear model to experimental data, where each method is meant to improve upon the one before. The first two methods are closely related. They have in common that learning is divided into two phases: the first phase estimates a value for β, and the second phase uses β to help estimate values for the in_i. The two methods differ in the optimization criteria they use to fit the model to the data. The third model is different from the first two in that it has only one learning phase, in which all parameters are estimated simultaneously. In this way, we hope to get a better fit to the data, since the estimate of β is now affected by how well the estimates of in_i fit the data, something that is impossible in the two-phase approach.

Recall that the linear model is given by equations of the form

$$y_{ij} \;=\; in_i \cdot (\mathbf{x}_{ij} \bullet \beta) \tag{4.2}$$

where the parameter vector β and all the in_i are unknown and must be learned. Of course, these equations are not exact, and provide at best an approximate description of the data. The goal is to see how closely they fit the data, and to estimate values for β and in_i in the process. From the discussion at the end of Section 4.2, we know it is only possible to estimate *relative* values for these quantities. This effectively means we can determine the *direction* of β but not its *magnitude*. In fact, in the absence of calibration data, the magnitude of β is meaningless. For this reason, the methods described in this section all impose constraints on the magnitude of β in order to obtain a unique solution.

LIN1: Two-Phase Learning

This approach factors out the amount of protein, in_i, from the set of Equations 4.2. The result is a set of linear eigenvector equations for the parameter vector β, which we can solve using standard eigenvector methods. With this estimate in hand, the value of each in_i can be estimated by linear regression.

From Equations 4.2, we see that protein i gives rise to the following set of equations, one equation for each peptide:

$$y_{i1} = in_i \cdot (\mathbf{x}_{i1} \bullet \beta) \qquad y_{i2} = in_i \cdot (\mathbf{x}_{i2} \bullet \beta) \qquad \cdots \qquad y_{in_i} = in_i \cdot (\mathbf{x}_{in_i} \bullet \beta) \quad (4.3)$$

Note that the unknown value in_i is the the same in each equation. Thus, by dividing each equation by the previous one, we can eliminate this unknown value, leaving the parameter vector β as the only unknown quantity. That is, $y_{ij}/y_{i,j-1} = (\mathbf{x}_{ij} \bullet \beta)/(\mathbf{x}_{i,j-1} \bullet \beta)$, for j from 2 to n_i. Cross multiplying gives $y_{ij}(\mathbf{x}_{i,j-1} \bullet \beta) = y_{i,j-1}(\mathbf{x}_{ij} \bullet \beta)$, and rearranging terms gives the following:[3]

$$\mathbf{z}_{ij} \bullet \beta = 0 \quad where \quad \mathbf{z}_{ij} = y_{ij}\mathbf{x}_{i,j-1} - y_{i,j-1}\mathbf{x}_{ij} \quad for\ j\ from\ 2\ to\ n_i,\ and\ i\ from\ 1\ to\ N. \quad (4.4)$$

Geometrically, these equations mean that the parameter vector β is orthogonal to each of the derived vectors $\mathbf{z}_{ij}$. Note that this is a constraint on the direction of β but not its magnitude, which is to be expected. Equation 4.4 is a restatement of Equation 4.2 with the unknown values in_i removed. Like Equation 4.2, it is an approximation, and our goal is to see how closely we can fit it to the data.

A simple approach is to choose β so that the values of $\mathbf{z}_{ij} \bullet \beta$ are as close to 0 as possible. That is, we can try to minimize the sum of their squares, $\sum_{i,j}(\mathbf{z}_{ij} \bullet \beta)^2$. Of course, this sum can be trivially minimized to 0 by setting $\beta = 0$. But, as described above, the magnitude of β is meaningless, and only its direction is important. So, without loss of generality, we minimize the sum of squares subject to the constraint that the magnitude of β is 1. To do this, we use the method of Lagrange multipliers. That is, we minimize the following function:

$$F(\beta, \lambda) = \sum_{i,j}(\mathbf{z}_{ij} \bullet \beta)^2 - \lambda(\|\beta\|^2 - 1) \quad (4.5)$$

[3]Of course, we could generate many more equations of this form by cross multiplying all possible pairs of equations from 4.3, instead of just the successive ones. However, only $n_i - 1$ of the resulting equations would be linearly independent.

Taking partial derivatives with respect to β and setting the result to 0, we get the equation

$$\sum_{ij} \mathbf{z}_{ij}(\mathbf{z}_{ij} \bullet \beta) \quad = \quad \lambda\beta \tag{4.6}$$

Taking the inner product of both sides with β gives $\sum_{ij}(\mathbf{z}_{ij} \bullet \beta)^2 \ = \ \lambda\beta \bullet \beta \ = \ \lambda\|\beta\|^2 \ = \ \lambda$, where the last equation follows from the constraint $\|\beta\| = 1$. We have therefore derived the following two equations:

$$\sum_{ij} \mathbf{z}_{ij}\mathbf{z}_{ij}^T \ \beta \ = \ \lambda\beta \qquad\qquad \lambda \ = \ \sum_{ij}(\mathbf{z}_{ij} \bullet \beta)^2$$

where the left equation is just Equation (4.6) expressed in matrix notation, with all vectors interpreted as column vectors. The left equation says that β is an eigenvector of the matrix $\sum_{ij} \mathbf{z}_{ij}\mathbf{z}_{ij}^T$, and the right equation says that its eigenvalue is just the sum of squares we want to minimize. We should therefore choose the eigenvector with the smallest eigenvalue.

Having estimated a value for β, we must now estimate values for the remaining unknowns, $in_1, \cdots, in_N$. Equations 4.3 suggests a natural way to do this. Letting $v_{ij} = \mathbf{x}_{ij} \bullet \beta$, we get $y_{ij} = in_i v_{ij}$, for j from 1 to n_i. Thus, for each value of i, we get a system of n_i linear equations involving in_i, and we can estimate the value of in_i using a simple univariate linear regression. The values of $in_1, \cdots, in_N$ can thus be estimated by carrying out N such regressions.

LIN2: Improving the Optimization Criterion

The previous section developed a method for fitting the linear model to data. The method uses the equation $\|\beta\| = 1$ to constrain the magnitude of the parameter vector β. However, the choice of this particular equation was arbitrary, since any equation of the form $\|A\beta\| = 1$ also constrains the magnitude of β, for any non-singular matrix A. In the previous section, we implicitly chose $A = I$, the identify matrix. In this section, we use $A = X$, where each row of matrix X is one of the feature vectors $\mathbf{x}_{ij}^T$. We still try to

minimize the sum of squares $\sum_{ij}(z_{ij} \bullet \beta)^2$, but subject to the new constraint $\|X\beta\| = 1$. Note that this constraint is equivalent to $\sum_{ij}(x_{ij} \bullet \beta)^2 = 1$.

The advantage of this constraint is that it leads to estimates of ionization efficiency and protein abundance that do not depend on arbitrary choices in data represention. For example, the estimates do not depend on whether we represent peptide mass in milligrams or micrograms. More generally, the estimates are invariant under any one-to-one linear transformation of the feature vectors. That is, suppose we let $\mathbf{x}'_{ij} = B\mathbf{x}_{ij}$, where B is a non-singular matrix. Then our estimates of protein abundance and ionization efficiency will not depend on whether we use $\mathbf{x}_{ij}$ or $\mathbf{x}'_{ij}$ to represent the peptides. This is how things should be. After all, the vectors $\mathbf{x}_{ij}$ and $\mathbf{x}'_{ij}$ contain exactly the same information, since each can be derived from the other. In fact, any change in the estimates would imply that the estimates are somewhat arbitrary, since they depend on arbitrary choices in data representation. In other words, if using $\mathbf{x}_{ij}$ leads to one set of estimates, while using $\mathbf{x}'_{ij}$ leads to another, then which estimates are correct? The estimation method developed here does not have this problem.

Before developing the method in detail, we first show that its estimates are indeed invariant under linear transformation. By Equation 4.2, it is enough to show that the values of $\mathbf{x}_{ij} \bullet \beta$ are invariant. To do this, first note that if $\mathbf{x}'_{ij} = A\mathbf{x}_{ij}$ and $\beta' = A^{-T}\beta$, then $\mathbf{x}_{ij} \bullet \beta = \mathbf{x}'_{ij} \bullet \beta'$. Thus, it is enough to show that if each feature vector, $\mathbf{x}_{ij}$, is replaced by $A\mathbf{x}_{ij}$, then the estimated parameter vector changes from $\hat{\beta}$ to $A^{-T}\hat{\beta}$. To show that the method behaves this way, first recall from Equation 4.4 that $\mathbf{z}_{ij} = y_{ij}\mathbf{x}_{i,j-1} - y_{i,j-1}\mathbf{x}_{ij}$. Thus $\mathbf{z}'_{ij} = y_{ij}\mathbf{x}'_{i,j-1} - y_{i,j-1}\mathbf{x}'_{ij} = A^{-T}(y_{ij}\mathbf{x}_{i,j-1} - y_{i,j-1}\mathbf{x}_{ij}) = A^{-T}\mathbf{z}_{ij}$. Thus, if $\beta' = A^{-T}\beta$, and $\mathbf{x}'_{ij} = A\mathbf{x}_{ij}$ for all i and j, then $\mathbf{z}_{ij} \bullet \beta = \mathbf{z}'_{ij} \bullet \beta'$ for all i and j. Hence,

$$\beta = \hat{\beta} \quad \text{minimizes} \quad \sum_{ij}(\mathbf{z}_{ij} \bullet \beta)^2 \quad \text{subject to} \quad \sum_{ij}(\mathbf{x}_{ij} \bullet \beta)^2 = 1$$

if and only if

$$\beta = A^{-T}\hat{\beta} \quad \text{minimizes} \quad \sum_{ij}(\mathbf{z}'_{ij} \bullet \beta)^2 \quad \text{subject to} \quad \sum_{ij}(\mathbf{x}'_{ij} \bullet \beta)^2 = 1$$

since none of the dot products changes value. Thus, all our estimates of ionization efficiency and protein abundance will remain unchanged under a one-to-one linear transaformation of the feature vectors. This is true for any method that solves this constrained minimization problem.

As in Section 4.2.1, the method developed here is based on Lagrange multipliers. That is, we minimize the following function, which is the new version of function (4.5):

$$F(\beta, \lambda) \quad = \quad \sum_{i,j} (\mathbf{z}_{ij} \bullet \beta)^2 \; - \; \lambda (\sum_{ij} (\mathbf{x}_{ij} \bullet \beta)^2 - 1)$$

Taking partial derivatives with respect to β and setting the result to 0, we now get

$$\sum_{ij} \mathbf{z}_{ij} (\mathbf{z}_{ij} \bullet \beta) \quad = \quad \lambda \sum_{ij} \mathbf{x}_{ij} (\mathbf{x}_{ij} \bullet \beta) \tag{4.7}$$

Taking the inner product of both sides with β, we get $\sum_{ij} (\mathbf{z}_{ij} \bullet \beta)^2 \; = \; \lambda \sum_{ij} (\mathbf{x}_{ij} \bullet \beta)^2 \; = \; \lambda$, where the last equation comes from the constraint $\sum_{ij} (\mathbf{x}_{ij} \bullet \beta)^2 \; = \; 1$. Thus, expressing Equation (4.7) in matrix notation, we have the following two equations:

$$\sum_{ij} \mathbf{z}_{ij} \mathbf{z}_{ij}^T \beta \quad = \quad \lambda \sum_{ij} \mathbf{x}_{ij} \mathbf{x}_{ij}^T \beta \qquad\qquad\qquad \lambda \; = \; \sum_{ij} (\mathbf{z}_{ij} \bullet \beta)^2$$

The left equation is a *generlalized* eigenvector equation, that is, an equation of the form $A\beta = \lambda B \beta$, where A and B are square matrices. In this case, since A and B are symmetric, the eigenvectors and eigenvalues are guaranteed to be real. There is therefore a smallest eigenvalue. As before, the right equation says that the eigenvalue is the sum of squares we are trying to minimize. We therefore choose the generalized eigenvector with the smallest eigenvalue. Estimating β in this way (the first phase of learning), we then use it to estimate values for the in_i using univariate linear regression, as before (the second phase of learning).

LIN3: Simultaneous Learning

Here, we outline a single-phase approach to learning, one that estimates values for β and all the in_i simultaneously. Since the estimate for one parameter takes into account the estimates for all the other parameters, we hope to get a better overall fit to the data.

In order to do this, we first transform Equation 4.2. Observe that the right-hand side of this equation contains a product of two unknowns, in_i and β. To eliminate this non-linear term, we divide both sides by in_i, to obtain a model that is linear in all the unknowns: $y_{ij}\alpha_i = \mathbf{x}_{ij} \bullet \beta$, where $\alpha_i = 1/in_i$. Since both sides of this equation contain unknown parameters, we cannot simply minimize the error between them, since by setting $\alpha_i = 0$ and $\beta = 0$ the error is trivially minimized to 0, which is clearly incorrect. Instead, we minimize the *angle* between two vectors, the vector of values on the right-hand side of the equation, and the vector of values on the left-hand side. Moreover, we do this by maximizing the cosine of the angle between them. In general, the cosine of the angle between two vectors, $\mathbf{v}$ and $\mathbf{w}$, is given by the formula $\mathbf{v} \bullet \mathbf{w}/\|\mathbf{v}\| \cdot \|\mathbf{w}\|$. We must therefore maximize the following expression:

$$\frac{\sum_{ij}(y_{ij}\alpha_i) \cdot (\mathbf{x}_{ij} \bullet \beta)}{\sqrt{\sum_{ij}(y_{ij}\alpha_i)^2}\sqrt{\sum_{ij}(\mathbf{x}_{ij} \bullet \beta)^2}} \tag{4.8}$$

Note that this measure of error is insensitive to the absolute magnitude of the parameters, α_i and β, which can all be scaled up or down by the same amount without affecting the angle between the two vectors.

In [7], we develop a method to efficiently carry out this maximization. It is based on Theorem 1 below, which is also proved in 5. In this theorem, Y_i is the column vector $(y_{i1}, y_{i2}, ..., y_{ik})^T$, and $\mathbf{X}_i$ is the matrix $(\mathbf{x}_{i1}, \mathbf{x}_{i2}, ..., \mathbf{x}_{in_i})^T$, where each $\mathbf{x}_{ij}$ is viewed as a column vector. They represent, respectively, the spectral peak intensities and peptide property vectors for protein i.

Theorem 1: *Expression (4.8) is maximized when the parameter vector β is a solution of the following generalized eigenvector equation:*

$$\rho^2 \sum_{ij} \mathbf{X}_i^T \mathbf{X}_i \; \beta \quad = \quad [\sum_i \; \mathbf{X}_i^T Y_i Y_i^T \mathbf{X}_i / \|Y_i\|^2] \; \beta$$

Moreover, it is the eigenvector corresponding to the largest eigenvalue, ρ^2. In addition,

$$\alpha_i \quad = \quad \mathbf{Y}_i^T \mathbf{X}_i^T \beta / \rho \|\mathbf{Y}_i\|^2$$

Finally, ρ is the maximum value of Expression (4.8). (So the minimum angle, θ, between the two vectors is given by $\rho = \cos(\theta)$).

4.2.2 Inverse Models

All of the methods developed above for fitting the linear model to data are easily adapted to fitting the inverse model. Recall that in the inverse model,

$$y_{ij} \;=\; in_i/(\mathbf{x} \bullet \beta)$$

By inverting both sides of the equation, we get,

$$y'_{ij} \;=\; in'_{ij} \cdot (\mathbf{x} \bullet \beta)$$

where $y'_{ij} = 1/y_{ij}$ and $in'_{ij} = 1/in_{ij}$. This is precisely the linear model. In addition, y'_{ij} is known and in'_{ij} is unknown, which is precisely the condition for applying the linear fitting methods developed in Section 4.2.1. When the linear methods LIN1, LIN2 and LIN3 are adapted in this way to the inverse model, we refer to them as INV1, INV2 and INV3.

4.2.3 Exponential Models

In this section, we develop two methods for fitting the exponential model to the data. Recall that in the exponential model,

$$y_{ij} \;=\; in_i \cdot e^{\mathbf{x}_{ij} \bullet \beta}$$

Taking logs of both sides converts this to a linear model:

$$\log y_{ij} \;=\; \log in_i + \mathbf{x}_{ij} \bullet \beta \tag{4.9}$$

The two methods described below differ in their treatment of the latent variables in_i. As with the linear methods of Section 4.2.1, one method is a two-phase learner, and the

other is single-phase. The first method operates in two phases: it estimates a value for the parameter vector, β, and then it uses this estimate to determine values for the protein input levels, in_i. In contrast, the second method operates in a single phase that estimates values for β and in_i simultaneously, finding the combination of values that best fits the data. Unlike the linear methods, the workhorse of these methods is linear regression, not eigenvector decomposition.

EXP1: Two-Phase Learning

From the above equations, we see that for the peptides from protein i,

$$\begin{cases} \log y_{i1} = \log in_i + \mathbf{x}_{i1}\beta \\ \log y_{i2} = \log in_i + \mathbf{x}_{i2}\beta \\ \qquad \cdots \\ \log y_{in_i} = \log in_i + \mathbf{x}_{in_i}\beta \end{cases} \tag{4.10}$$

¿From this, we get that $\log y_{i,j-1} - \log y_{i,j} = (\mathbf{x}_{i,j-1} - \mathbf{x}_{i,j})\beta$ for j from 2 to n_i, or equivalently $Y_{ij} = X_{ij} \bullet \beta$, where

$$Y_{ij} = \log y_{i,j-1} - \log y_{i,j} \qquad X_{ij} = \mathbf{x}_{i,j-1} - \mathbf{x}_{i,j} \tag{4.11}$$

β can thus be estimated from the new variables X_{ij} and Y_{ij} using linear regression. This is the first phase of learning. Once β is known, its value can be used to help estimate values for the remaining unknowns, $in_1, ..., in_N$. This is the second phase of learning. Equation 4.9 suggests a natural approach. Letting $v_{ij} = \log y_{ij} - \mathbf{x}_{ij} \bullet \beta$, we get $v_{ij} = \log in_i$, for j from 1 to n_i. Like Equation 4.9, this equation is only approximate. The value of $\log in_i$ that minimizes the mean squared error is just the mean of the v_{ij}. We therefore use $\log in_i = (v_{i1} + \cdots + v_{in_i})/n_i$.

Although it is straightforward, a potential problem with this method is in the error that it minimizes. We would like to minimize (the sum of squares of) the errors $\varepsilon_{ij} = logy_{ij} - \log in_i - \mathbf{x}_{ij} \bullet \beta$. Instead, however, the method minimizes (the

sum of squares of) $\varepsilon_{ij} - \varepsilon_{i,j-1}.$, which is not the same thing. The method developed next solves this problem.

EXP2: Simultaneous Learning

In order to minimize the errors ε_{ij} directly, we treat the unknown values $\log in_i$ in Equation (4.9) as regression parameters. To this end, we define β' to be the extended parameter vector $(\beta_1, ..., \beta_p, b_1, ..., b_N)^T$. Here, b_i is a parameter representing $\log in_i$, and $(\beta_1, ..., \beta_p)^T$ is the original parameter vector, β, used above. Values for all the parameters in β' will be estimated simultaneously in a single act of linear regression. To do this, we define a number of matrices. First, we define $\mathbf{O}$ to be the following sparse matrix:

$$\left(\begin{array}{ccc|ccc|ccc|ccc} 1 & \cdots & 1 & 0 & \cdots & 0 & \cdots & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 1 & \cdots & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots & & \ddots & & \vdots & \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & \cdots & 1 & \cdots & 1 \end{array}\right)^T$$

where the i^{th} vertical block has n_i columns. In this matrix, each row corresponds to a protein, and each column to a peptide. A 1 in the matrix means that the peptide belongs to the protein. In addition, we define $\mathbf{X}$ to be the matrix of predictors $(\mathbf{x}_{11}, ..., \mathbf{x}_{1,n_1}, \mathbf{x}_{21}, ..., \mathbf{x}_{2n_2}, ..., \mathbf{x}_{N1}, ..., \mathbf{x}_{Nn_N})^T$, and Y to be the column vector of responses $(Y_{11}, ..., Y_{1,n_1}, Y_{21}, ..., Y_{2n_2}, ..., Y_{N1}, ..., Y_{Nn_N})^T$, where $Y_{ij} = \log y_{ij}$. With this notation, it is not hard to see that the set of linear Equations (4.9) becomes the single matrix equation $Y = \mathbf{V}\beta'$ where $\mathbf{V}$ is the extended predictor matrix $(\mathbf{X}, \mathbf{O})$. Thus, the problem again reduces to linear regression, though with a far larger set of parameters. The solution is

$$\beta' = (\mathbf{V}^T\mathbf{V})^{-1}\mathbf{V}^T Y \tag{4.12}$$

Observe that $\mathbf{X}$ is a $M \times p$ matrix, where $M = \sum_i n_i$ is the total number of peptides,

and p is the number of features in the vectors $\mathbf{x}_{ij}$. Likewise, $\mathbf{O}$ is a $M \times N$ matrix, where N is the number of proteins. Consequently, $\mathbf{V}$ has dimensions $M \times (p+N)$, and $\mathbf{V}^T\mathbf{V}$ has dimensions $(p+N) \times (p+N)$. Although p is relatively small in our datasets, N is large, so the matrix $\mathbf{V}^T\mathbf{V}$ is extremely large, and inverting it is computationally very expensive.

Fortunately, because of the sparse and regular structure of $\mathbf{O}$, we can develop a more efficient procedure. The first step is to partition the matrix $(\mathbf{V}^T\mathbf{V})^{-1}$ into blocks as follows:

$$(\mathbf{V}^T\mathbf{V})^{-1} \quad = \quad \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) \tag{4.13}$$

Here, the submatrices have the following dimensions: A is $p \times p$, B is $p \times N$, C is $N \times p$, and D is $N \times N$. The submatrix A is therefore small, while D is extremely large. The next step is to solve for A, B, C and D, as follows:

$$I \;=\; (\mathbf{V}^T\mathbf{V})(\mathbf{V}^T\mathbf{V})^{-1} \;=\; \left(\frac{\mathbf{X}^T}{\mathbf{O}^T}\right)(\mathbf{X},\mathbf{O})\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) \;=\; \left(\begin{array}{c|c} \mathbf{X}^T\mathbf{X}A + \mathbf{X}^T\mathbf{O}C & \mathbf{X}^T\mathbf{X}B + \mathbf{X}^T\mathbf{O}D \\ \hline \mathbf{O}^T\mathbf{X}A + \mathbf{O}^T\mathbf{O}C & \mathbf{O}^T\mathbf{X}B + \mathbf{O}^T\mathbf{O}D \end{array}\right)$$

We can view this as four matrix equations in four unknowns, A, B, C, D. Solving, we get

$$A \;=\; (\mathbf{X}^T\mathbf{X} - \hat{\mathbf{X}}^T P^{-1}\hat{\mathbf{X}})^{-1} \qquad B \;=\; -A\tilde{\mathbf{X}}^T \qquad C \;=\; B^T \qquad D \;=\; P^{-1} - \tilde{\mathbf{X}}A\tilde{\mathbf{X}}^T$$

where $\hat{\mathbf{X}} = \mathbf{O}^T\mathbf{X}$, $P = \mathbf{O}^T\mathbf{O}$, and $\tilde{\mathbf{X}} = P^{-1}\hat{\mathbf{X}}$. The first point to notice here is that A is a small matrix, of size $p \times p$, so even though it requires a matrix inversion, it is not costly. The second point is that the matrices P and $\mathbf{O}$ are both exremely large, of size $N \times N$ and $M \times N$, respectively. However, neither one needs to be materialized. To see this, note that the rows of $\mathbf{O}^T$ are orthogonal, and P is therefore diagonal. In fact, the i^{th} diagonal element of P is simply n_i, the number of peptides in protein i. The i^{th} diagonal element of P^{-1} is therefore $1/n_i$. Multiplication by P or P^{-1} is therefore a simple operation. Likewise for left multiplication by $\mathbf{O}^T$. For instance, the i^{th} row of $\hat{\mathbf{X}} = \mathbf{O}^T\mathbf{X}$ is simply $\sum_j \mathbf{x}_{ij}^T$. In this way, neither P, P^{-1} nor $\mathbf{O}$ needs to be materialized.

Neither does the extremely large matrix D. In fact, the largest matrix that needs to be materialized is the data matrix, $\mathbf{X}$. To see this, first rewrite Equation 4.12 as

$$\left(\begin{array}{c}\beta\\\hline \mathbf{b}\end{array}\right) = \left(\begin{array}{c|c}A & B\\\hline C & D\end{array}\right)\left(\begin{array}{c}\mathbf{X}^T\\\hline \mathbf{O}^T\end{array}\right)Y = \left(\begin{array}{c}A\mathbf{X}^TY + B\mathbf{O}^TY\\\hline C\mathbf{X}^TY + D\mathbf{O}^TY\end{array}\right)$$

where we have decomposed the extended parameter vector, β', into the original parameter vector, $\beta = (\beta_1, ..., \beta_p)^T$, and a vector of the new parameters, $b = (b_1, ..., b_N)^T$. Expanding the definition of D, we get two equations:

$$\beta = A\mathbf{X}^TY + B\hat{Y} \tag{4.14}$$

$$\begin{aligned} b &= C\mathbf{X}^TY + (P^{-1} - \tilde{\mathbf{X}}A\tilde{\mathbf{X}}^T)\hat{Y} \\ &= C\mathbf{X}^TY + \tilde{Y} - \tilde{\mathbf{X}}A\tilde{\mathbf{X}}^T\hat{Y} \end{aligned} \tag{4.15}$$

where $\hat{Y} = \mathbf{O}^TY$ and $\tilde{Y} = P^{-1}\hat{Y}$. Note that $\hat{Y}$ is a column vector whose i^{th} component is $\hat{Y}_i = \sum_j Y_{ij}$. Likewise, $\tilde{Y}$ is a column vector whose i^{th} component is $\tilde{Y}_i = \hat{Y}_i/n_i$. Thus, it is still unneccessary to materialize the very large matrices P, P^{-1} and $\mathbf{O}$.

Finally, observe that the largest matrix in Equations 4.14 and 4.15 is the data matrix $\mathbf{X}$, which has size $M \times p$. The matrices $\hat{\mathbf{X}}$ and $\tilde{\mathbf{X}}$ both have size $N \times p$, the matrix A has size $p \times p$, and the matrices B and C have sizes $p \times N$ and $N \times p$, respectively. All of these are smaller than $\mathbf{X}$ since $p < N < M$. Moreover, if all the matrix multiplications are carried out from right to left, then all the intermediate results are column vectors, not large matrices. In this way, the parameter vectors β and b can be estimated without having to materialize or manipulate any matrices larger that $\mathbf{X}$. In fact, for fixed p, the total cost of estimating the parameters is linear in M, the number of peptides; *i.e.*, the total cost is linear in the number of data points.

4.3 Experiments

We evaluated the methods and models developed above on real and simulated datasets. This section describes the design of the experiments, and the evaluation methods. The

main difficulty in evaluating the methods is the distribution of the real data. As shown in Chapter 2, it ranges over several orders of magnitude and is highly skewed, with most data concentrated at very low values. We deal with these difficulties in two ways. First, we use the Spearman rank correlation coefficient to measure the goodness of fit of our estimates to the observed values [3]. Unlike the more common Pearson correlation coefficient, which measures *linear* correlation, Spearman's coefficient measures *monotone* correlation and is insensitive to extreme data values. In addition, we use log-log plots of observed v.s. estimated values to provide an informative visualization of the fit.

4.3.1 Simulated Datasets

In addition to real-world datasets (described above), we generated simulated data on which to test the learning models and methods developed in Sections 4.1 and 4.2. While real-world data tests the biological relevance of our methods, simulated data allows us to test their mathematical and computational correctness. To this end, we use simulated data that is based on the same statistical models as our learning methods. This serves two purposes. First, it acts as a sanity check, since errors in either the mathematics or the programming of the methods can easily appear as unexpected or bizarre results. Second, it tells us the performance we can expect from the methods under ideal circumstances, and with controllable amounts of noise. In [6], a thorough set of simulated experiments is performed. In this paper, we present a small sample of results to demonstrate that our methods can in principle estimate the amounts of protein at the input to the mass spectrometer.

The simulator generates data for the linear, inverse and exponential models. Although different, these models each have the same kinds of parameters: an input amount, in_i, for each protein; a feature vector, $\mathbf{x}_{ij}$, for each peptide; and a parameter vector, β, with which to compute ionization efficiencies. Protein amounts are generated randomly from a uniform distribution ranging from 20 to 250. Peptides are represented by vectors

with 22 components, as in the real datasets. For each component of a peptide vector, values are generated randomly from a normal distribution with mean 4 and standard deviation 2. For each component of the vector β, values are generated randomly from a uniform distribution ranging from 0 to 1. From this data, we compute ionization efficiencies and spectral count values, according to our models of MS/MS data. In each model, we add noise to $\mathbf{x}_{ij} \bullet \beta$, since this is what our methods minimize. We then compute ionization efficiencies from these values. Thus, for the linear model, the ionization efficiency of peptide i of protein j is $ie_{ij} = \mathbf{x}_{ij} \bullet \beta + noise$, for the inverse model it is $ie_{ij} = 1/(\mathbf{x}_{ij} \bullet \beta + noise)$, and for the exponential model it is $ie_{ij} = e^{\mathbf{x}_{ij} \bullet \beta + noise}$. In all these cases, the noise is generated from a normal distribution with mean 0. Finally, spectral count values are generated, and in all three models, the spectral count of peptide ij is $y_{ij} = in_i \cdot ie_{ij}$. For the simulated data, the standard deviation of the noise is constant, but different experiments use different amounts of noise, *i.e.*, noise with different standard deviations. Here, noise level is measured with respect to the variation within the peptide data. For example, a noise level of 50% means that the standard deviation of the noise is 50% of the standard deviation of the components of the peptide vectors $\mathbf{x}_{ij}$. For each simulated experiment, we simulated 100 different proteins, each fragmenting into 10 different peptides, for a total of 1000 peptides. In addition, in order to obtain a nice visual representation of the accuracy of our estimates of protein level, we divided the proteins into ten groups, where each protein in a group has exactly the same input amount, in_i.

Each simulated experiment uses the same protein levels, in_i, the same peptide feature vectors, $\mathbf{x}_{ij}$, the same parameter vector, β, and often the same noise. Of course, the datasets for the three models—linear, inverse, and exponential— will still differ, since, according to the formulas given above, the ionization efficiencies, ie_{ij}, and spectral count, y_{ij}, will be different in the three models.

4.3.2 Experimental Design

As described in Section 4.1, only those proteins that produce at least two peptides with positive spectral counts are useful for fitting models to data. We first identified these proteins. We then built an index for these proteins, and a separate index for their peptides. For the Brain dataset, the indexes contain 8,527 peptides and 1,664 proteins, respectively. For the Heart dataset, the indexes contain 7,660 peptides and 1,281 proteins, respectively. For the Kidney dataset, the indexes contain 7,074 peptides and 1,291 proteins, respectively. In addition, in order to estimate the generalization error of the fitted models, each of these three data sets is divided randomly into 10 subsets, we use 10-fold cross-validation to estimate the standard deviation and mean of the performance of each model.

Finally, to use the models developed in Section 4.1, each peptide must also be represented as a vector, $\mathbf{x}$. we evaluate several ways of doing this, using vectors with 22, 42, 62 and 232 components, respectively. The 22-component vector represents the amino-acid composition of a peptide. Recall that a peptide is a sequence of amino acids. Since there are twenty different amino acids, the vector has 20 components, $(x_1, ..., x_{20})$, where the value of x_i is the number of occurrences of a particular amino acid in the peptide. In addition, the vector has a 21^{st} component, x_0, whose value is always 1, to represent a bias term, as is common in linear regression [24]. The vector also has a 22^{nd} component whose value is the charge of the peptide ion, as given in Table 2.1 in Chapter 2. The 232-component vector uses these same 22 components plus an additional 210 quadratic components of the form $x_i x_j$ computed from x_i and x_j for $1 \leq i, j \leq 20$.

These vectors capture the charge and amino-acid composition of a peptide, but they do not contain any sequential information. The 42-component vector ameliorates this situation somewhat. It divides a peptide sequence into two subsequences, by cutting it in half, and represents the amino-acid composition of each half. This requires 40 vector components. Again, we add a 41^{st} component to act as a bias term, and a 42^{nd}

component to represent charge. The 62-component vector carries this idea one step further by dividing a peptide into three subsequences, thereby capturing more sequential information.

These four different peptide representations can be used with any of the eight fitting methods developed in Section 4.2, namely, LIN1, LIN2, LIN3, EXP1, EXP2, INV1, INV2 and INV3. When applied to the training data, these methods each estimate a vector of parameters, β, from which we can estimate the ionization efficiency of any peptide. As described in Section 4.1, the ionization efficiency of a peptide is given by the following formulas, depending on the model:

$$\text{Linear}: \; ie = \mathbf{x} \bullet \beta \qquad \text{Exponential}: \; ie = e^{\mathbf{x} \bullet \beta} \qquad \text{Inverse}: \; ie = 1/(\mathbf{x} \bullet \beta) \tag{4.16}$$

where $\mathbf{x}$ is the vector representation of the peptide, as described above. To estimate the accuracy of these models, we use them to predict the spectral counts of all the peptides in the testing data. We then compare the predictions to the observed values.

The first step is to estimate the amount of each protein in the testing data. Recall that $y_{ij} = in_i \cdot ie_{ij}$, where in_i is the amount of protein i, ie_{ij} is the ionization efficiency of peptide ij, and y_{ij} is the value of the peptide's spectral count. The goal is to estimate a value for in_i given the observed values of y_{ij} and the estimated values of ie_{ij}. There are numerous ways this can be done, and the most appropriate depends on the statistical model being used. Recall that in each model, the formula $y_{ij} = in_i \cdot ie_{ij}$ is transformed so that $\mathbf{X} \bullet \beta$ appears as a linear term. In particular:

$$\text{Linear}: \; y_{ij} = in_i \cdot (\mathbf{x}_{ij} \bullet \beta) \qquad \text{Exponential}: \; \log(y_{ij}) = \log(in_i) + \mathbf{x}_{ij} \bullet \beta$$

$$\text{Inverse}: \; y_{ij}^{-1} = in_i^{-1} \cdot (\mathbf{x}_{ij} \bullet \beta)$$

In each of these models, y_{ij}, $\mathbf{x}_{ij}$ and β are known, and in_i must be estimated. Thus, in the linear model, in_i can be estimated by univariate linear regression. Likewise, in the inverse model, after first computing values for y_{ij}^{-1}, a value for in_i^{-1} can be estimated

by univariate linear regression, from which the value of in_i can be estimated as $1/in_i^{-1}$. Finally, in the exponential model, a value for $\log(in_i)$ can be estimated as the mean of $\log(y_{ij}) - (\mathbf{x}_{ij} \bullet \beta)$, from which the value of in_i can be estimated by taking exponentials.

Let $\hat{in}_i$ denote the estimate of in_i, and let $\hat{ie}_{ij}$ be the estimate of ie_{ij}. If these were the true amount of protein i and the true ionization efficiency of peptide ij, then the value of the peptide's spectral count would be $\hat{y}_{ij} = \hat{in}_i \times \hat{ie}_{ij}$. By comparing this estimate to the observed value, y_{ij}, for each peptide in the test set, we obtain an estimate of the generalization error of our methods.

A common way to make such a comparisons is to use the Pearson correlation coefficient, which measures the linear correlation between two random variables [3]. However, because of the distribution of our data—highly skewed and ranging over several orders of magnitude—the Pearson correlation coefficient can be misleading, since its value is dominated by a relatively small number of extremely large data values. In addition, because of the variety of models we use, it is not clear whether we should compute the correlation of y v.s. $\hat{y}$, or of $\log(y)$ v.s. $\log(\hat{y})$, or of $1/y$ v.s. $1/\hat{y}$. Fortunately, all these problems are solved by using the *Spearman* rank correlation coefficient [3]. This is similar to the Pearson correlation coefficient, except that instead of correlating the data values themselves, it correlates their ranks. It is therefore insensitive to the exact values of the data, and in particular, to extremely large values. Moreover, instead of measuring linear correlation, it measures monotone correlation (*i.e.*, the tendency of one variable to increase or decrease with the other variable). In fact, the value of the Spearman rank correlation coefficient is invariant under monotonic transformations of the data. Thus, it does ot matter whether we consider y or $\log(y)$ or $1/y$.

Finally, in order to judge the significance of our results, we compare them to a set of naive and random models. We use naive versions of the linear, exponential and inverse models, denoted LIND, EXPD and INVD, respectively (where D stands for "Dumb").

In each naive model, each peptide is simply assumed to have an ionization efficiency of 1. From Equations (4.16), this is equivalent to assuming that, for all peptides, $\mathbf{x} \bullet \beta$ is 1 in the linear model, 0 in the exponential model, and 1 in the inverse model. We then specialize the methods described above for estimating in_i, as follows:

- in the linear model, in_i is estimated to be the mean of y_{ij};
- in the exponential model, $\log(in_i)$ is estimated to be the mean of $\log(y_{ij})$;
- in the inverse model, $1/in_i$ is estimated to be the mean of $1/y_{ij}$.

Using these estimates of protein input, the spectral count of each peptide is estimated to be $\hat{y}_{ij} = \hat{in}_i \cdot \hat{ie}_{ij} = \hat{in}_i$, since $\hat{ie}_{ij} = 1$ in all of the naive models. Finally, we compare these estimated spectral counts to the observed values using Spearman's rank correlation coefficient. Note that for the naive models, the Spearman coefficient will not depend on which vectors we use to represent peptide sequences, since the models themselves are independent of this. Similarly, we propose random versions of the linear, exponential and inverse models, denoted LINR, EXPR and INVR, respectively (where R stands for "Random"). In each random model, each peptide is simply assumed to have an ionization efficiency which satisfies uniform distribution $U[0, 1]$, the model fitting of the random models are exactly the same as the naive models.

4.4 Results and Analysis

4.4.1 Experiments on Simulated Data

Since the protein levels are unknown in the real datasets, we must resort to simulated data in order to directly compare the estimated protein levels to their true values. Figures 4.1, 4.2 and 4.3 illustrate the estimation of protein input levels for the simulated data. The straight blue lines in the figures represent the true protein levels, while the jagged red

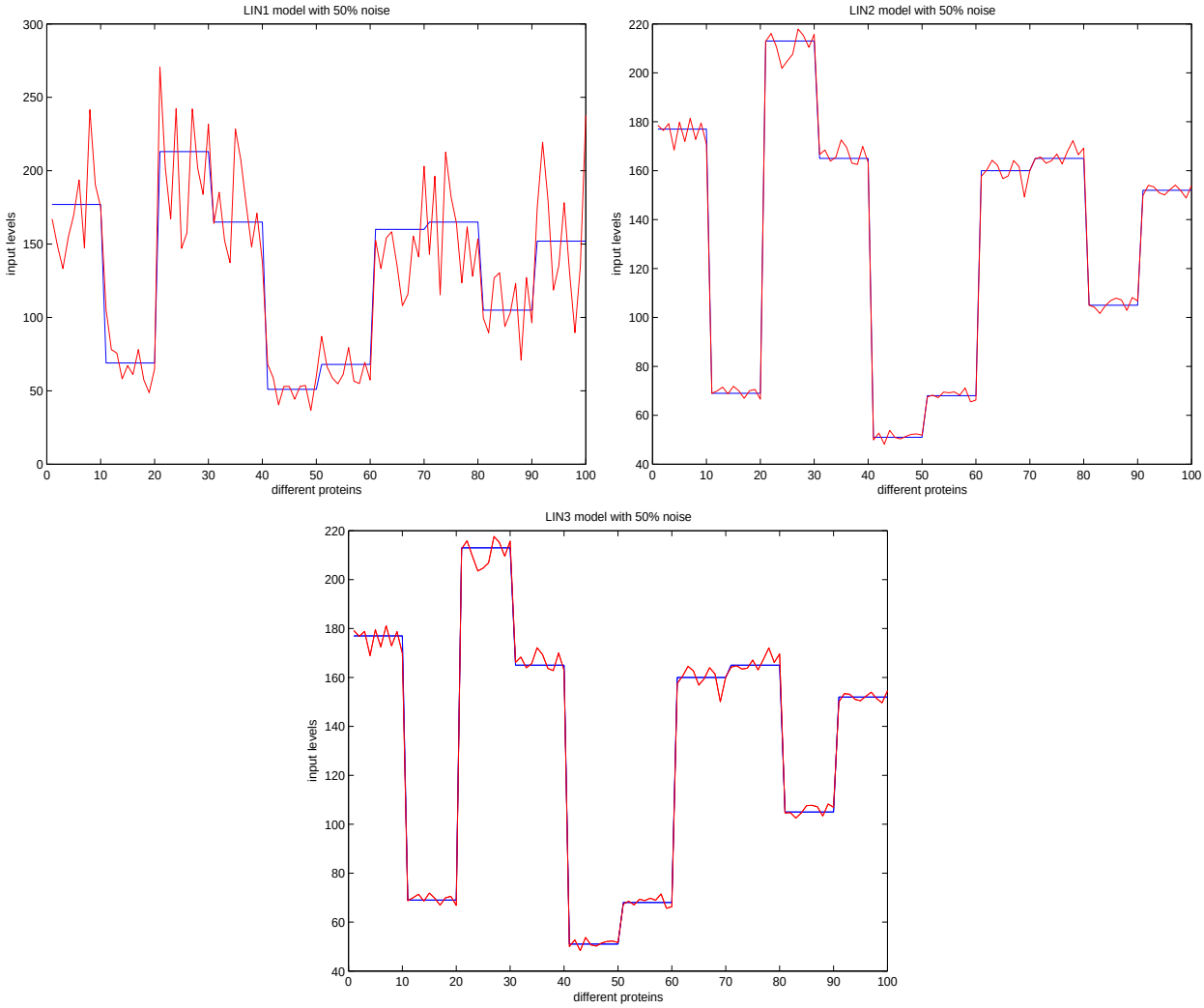

Figure 4.1: Illustration of the estimated input levels for different linear models with noise level 50%, The figures shows the fitting result for LIN1,LIN2,LIN3,

lines represent their estimated values.[4] Notice that the curve of estimated values has the same general shape as the curve of real values: when the real values rise or fall dramatically, so to the estimated values; and when the real values remain constant, the estimated values vary around the true value. However, the amount of variation varies considerably from model to model.

[4]As emphasized in Sections 4.1 and 4.2, it is only possible to estimate the relative values of protein levels, not their absolute values. Thus, the true and estimated protein levels may differ by a possibly-large constant factor. We have compensated for this in the figures by scaling the estimates so that they have the same sum of squares as the true values.

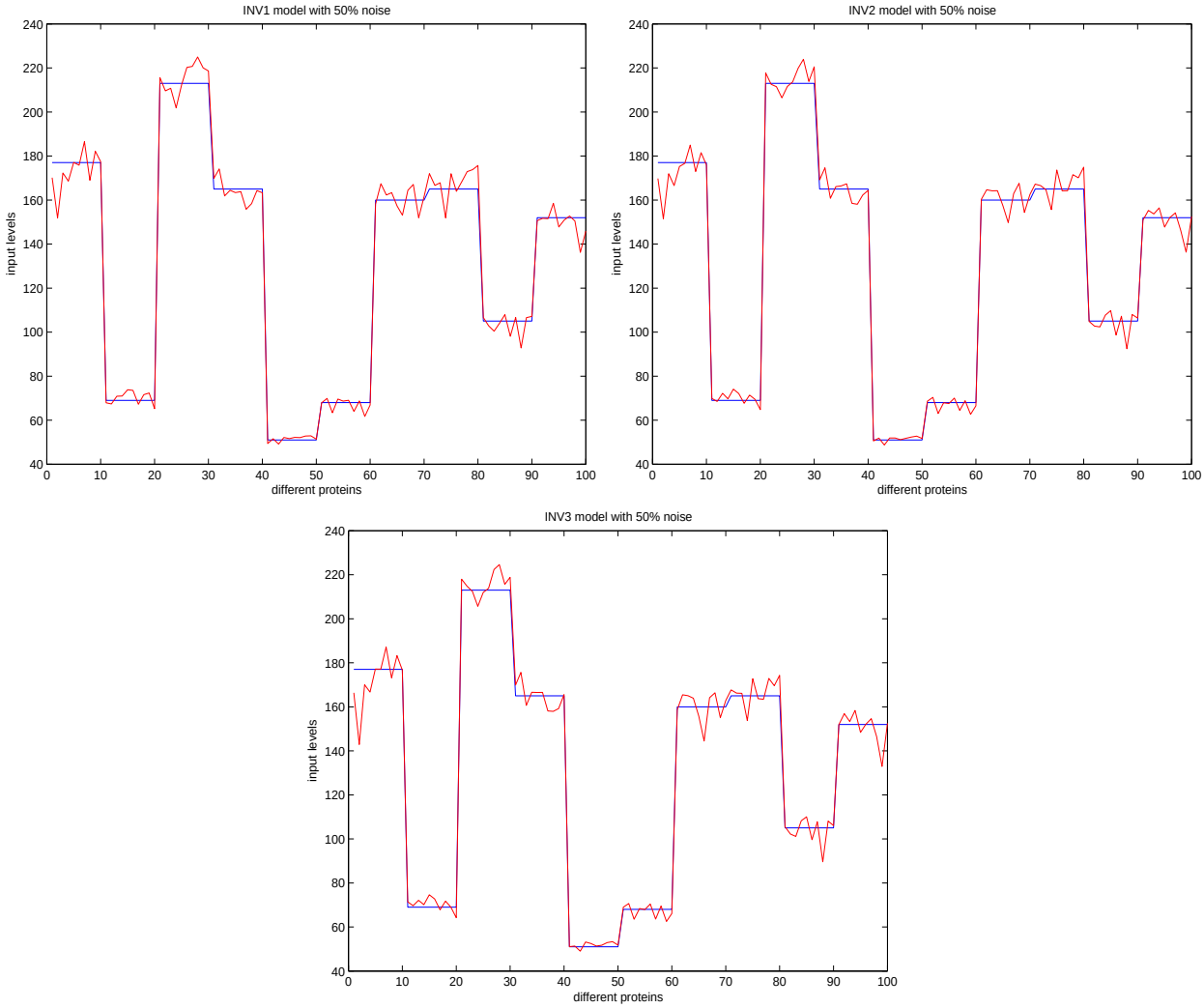

Figure 4.2: Illustration of the estimated input levels for different inverse models with noise level 50%, here shows the results for INV1, INV2 and INV3

Figure 4.1 clearly show that the variance in the estimated protein levels is much greater for LIN1 than for the other two linear models, LIN2 and LIN3, which have about equal variance. In addition, figure 4.2 show that the variance is about equal for all three inverse models, INV1, INV2 and INV3. The upper and lower panels cannot be directly compared, however, since the linear and inverse models are based on different noise models. (As described in Section 4.3.1, in the linear model, noise is added to ionization efficiency, ie, while in the inverse model, noise is added to $1/ie$.)

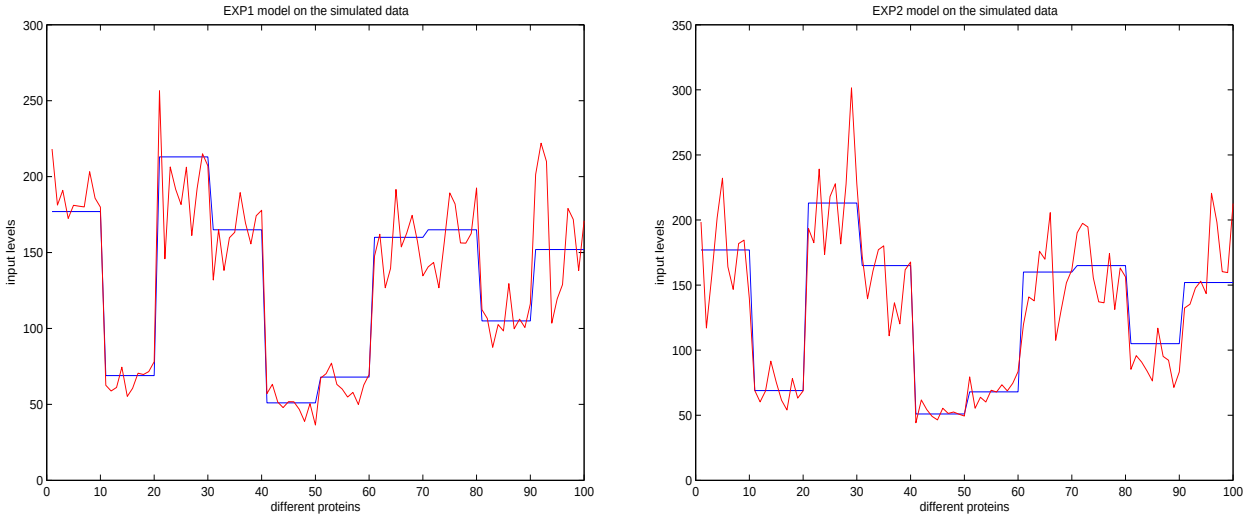

Figure 4.3: Illustration of the estimated input levels for different exponential models with noise level 25%. The left panel shows results for EXP1, while the right panel shows results for EXP2

Figure 4.3 shows a similar trend for the two exponential models, EXP1 and EXP2: the curve of estimated protein levels has the same general shape as the curve of true protein levels. In addition, we can see that the variance in the estimated protein levels is about the same for the two exponential models. Note, however, that the ostensible noise level is half of that used in Figures 4.1 and 4.2 for the linear and inverse models, yet the variance in protein estimates appears to be much higher. This is because the noise model is multiplicative, not additive. In particular, as described in Section 4.3.1, the ionization efficiency is multiplied by e^{noise}. Thus, an ostensible noise level of 25% has a much greater impact on the exponential model than on the linear model.

4.4.2 Experiments on Real-World Data

Tables 4.1, 4.2 and 4.3 show mean testing Spearman rank correlation coefficients of 10-fold cross validation. While tables 4.4 to 4.11 are complete results for 10-fold cross validation for mouse brain dataset, in these tables, we collect the mean of both the training and

testing cases with the corresponding standard deviations, the entries are Spearman rank correlation coefficients between y and $\hat{y}$ or ie vs. $\hat{ie}$. For more results on Mouse Heart dataset and Mouse Kidney dataset, please refer to the Appendix. In our experiment, according to the 10-fold cross validation, we divide the dataset into 10 parts, each time, we use 9 partitions among them to be the training set, while the remaining 1 as the testing set, the Spearman correlation coefficients of spectral count y vs. estimated value $\hat{y}$ for the 10 runs are calculated. The mean values and standard deviations on both training cases and testing cases are recorded to evaluate the performance of different methods. For every combination of eleven fitting methods, four peptide representations, and three Mouse datasets. The rows of each table represent fitting methods, and the columns represent vector representations of peptides (identified by the number of features in the vectors.) Each table entry is the mean value Spearman rank correlation coefficients of y and $\hat{y}$ for 10-fold cross validation, that is, of observed peptide spectral counts and estimated values. The performance of the these methods, along with the naive methods, is illustrated graphically in Figure 4.4, 4.5, 4.6, which plot the mean value of Spearman rank correlation coefficients of 10-fold cross validation against the size of the feature vectors. Each point in the curve represents the mean Spearman correlation coefficient, while each curve in the figure represents a different method.

The following observations about the fitting methods are immediately apparent from these tables: LIN1 and LIN2 perform the worst; of the linear methods, LIN3 performs the best; of the inverse methods, INV3 performs the best; INV3 always performs better than LIN3; the exponential methods perform the best of all; the naive methods perform better than many of the other methods; and of the naive methods, INVD performs the worst; Except the models LIN1 and LIN2, INV1, all the models perform better than random models.

Recall that the only difference between LIN1 ad LIN2 is the constraints on which they are based. The poorer performance of LIN1 suggests that its constraint is much less

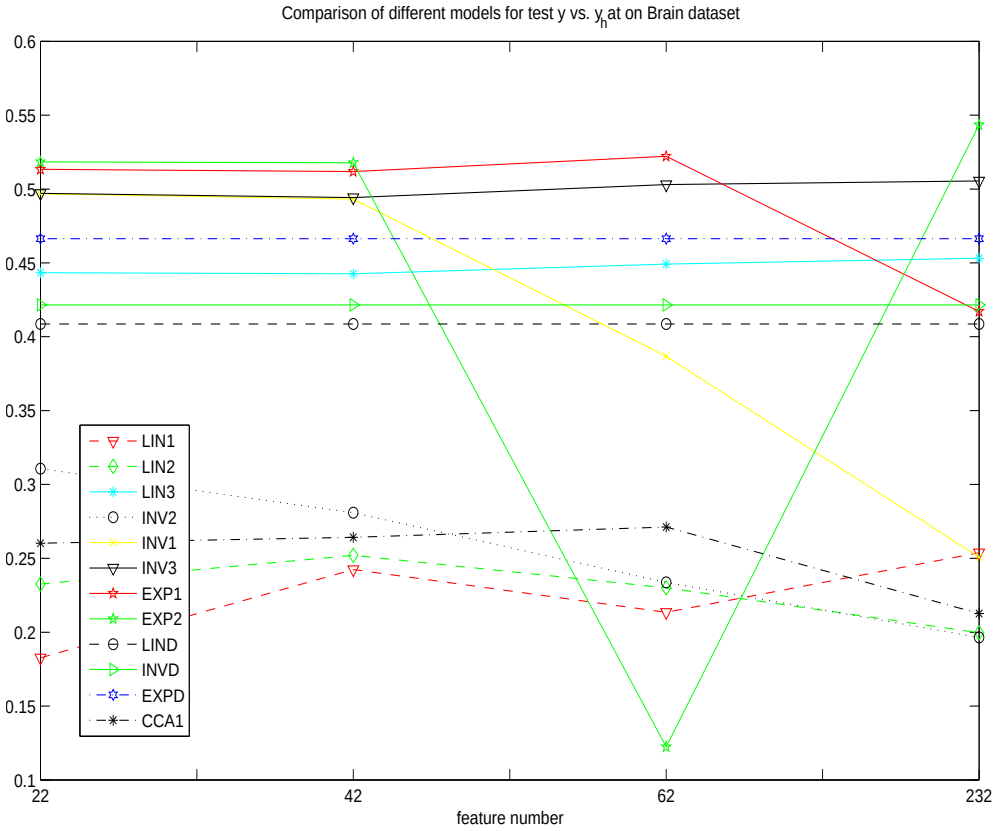

Figure 4.4: A comparison of the generalization performance of different models on the Mouse Brain dataset

appropriate than the constraint for LIN2, as argued in Section 4.2.1. Recall also that LIN1 and LIN2 use a two-phase approach to learning, in which the parameter vector β is estimated first, after which protein input levels are estimated from β. In contrast, LIN3 is based on a single-phase approach to learning, in which parameters and protein levels are all learned simultaneously. The better performance of LIN3 over LIN1 and LIN2 on the real datasets suggests that the single-phase approach is more appropriate, at least for tandem mass spectrometry data, as suggested in Section 4.2.1. (Interestingly, in our experiments on simulated data, LIN1 and LIN2 performed comparably, and clearly worse than LIN1.) These conclusions are all corroborated by the performance of INV1, INV2

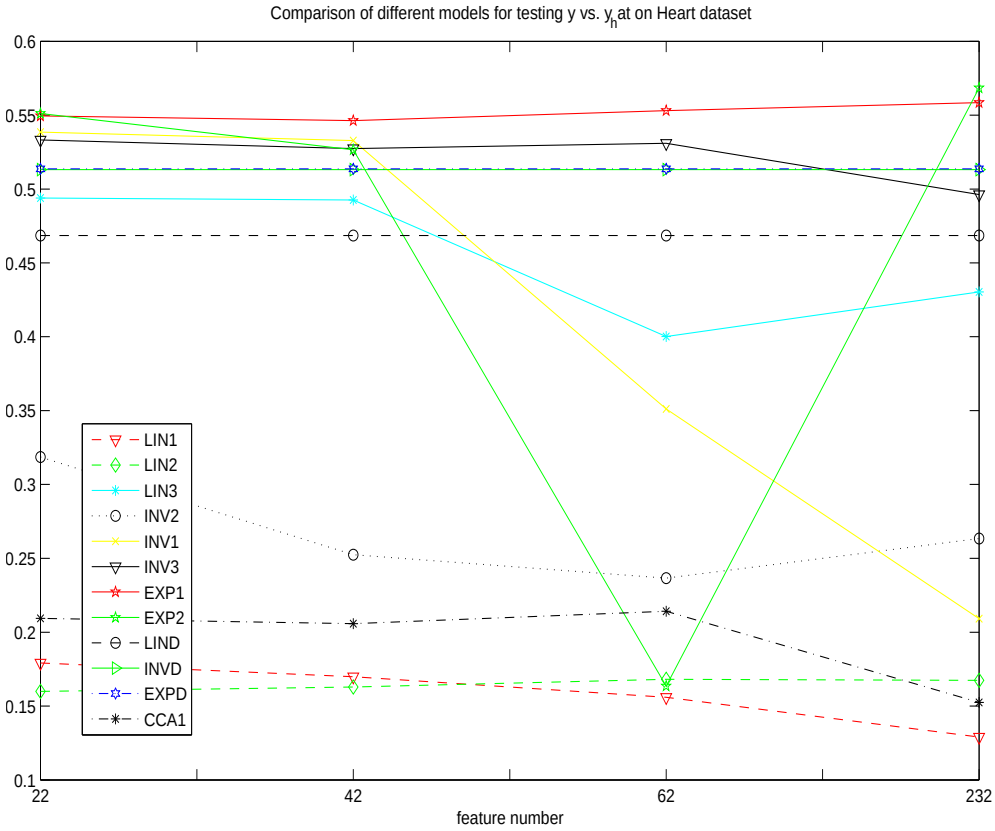

Figure 4.5: A comparison of the generalization performance of different models on the Mouse Heart dataset

and INV3, which parallels that of LIN1, LIN2 ad LIN3, upon which they are based.

That INV3 always performs better than LIN3 suggests that the inverse transformation upon which INV3 is based had its intended effect. However, the result is not conclusive, since the naive inverse method, INVD, does not perform significantly better than the other two naive methods, LIND and EXPD, especially on the kidney dataset. In the naive methods, there is no attempt to model the ionization efficiency of peptides, so that data transformation is the only factor that distinguishes them. The random models be be viewed as a benchmark, the poor performance of these random models suggests that our proposed modeling strategy is effective.

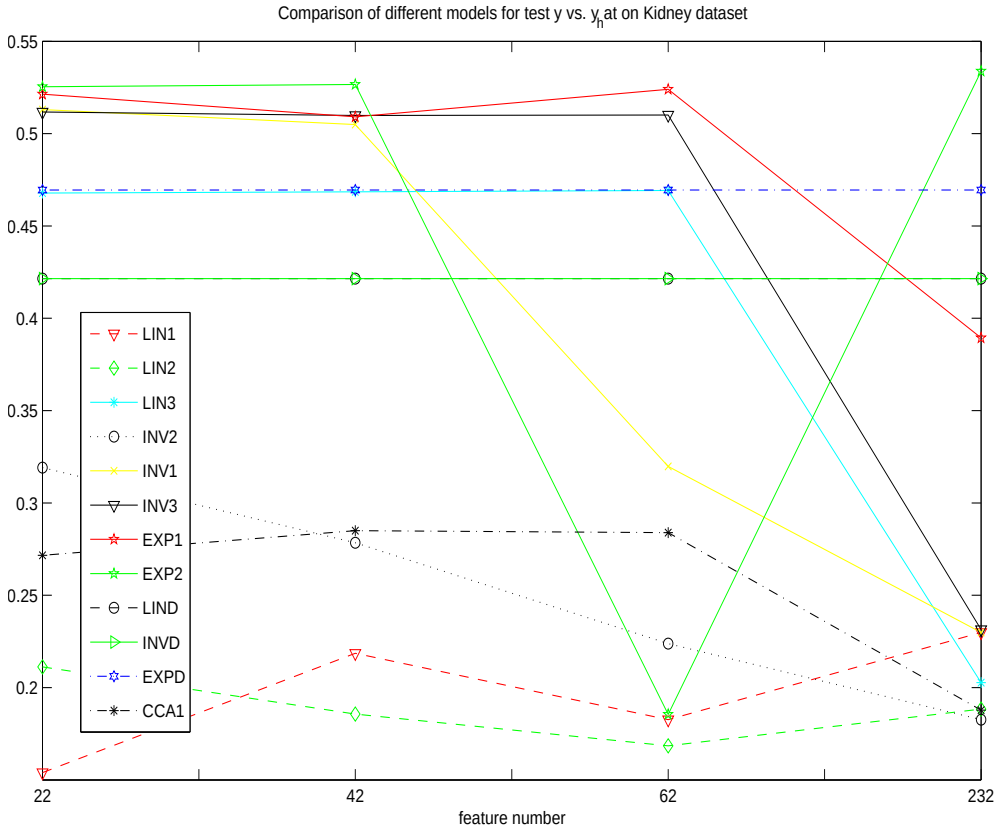

Figure 4.6: A comparison of the generalization performance of different models on the Mouse Kidney dataset

The superior performance of the exponential models suggests that the logarithmic transformation on which they are based is the most appropriate. This is perhaps not surprising. Logarithms remove the problem of data ranging over several orders of magnitude, while simultaneously compressing a sparse set of extremely large values into a denser set of smaller values. This effectively deals with two of the main problems in our datasets. In addition, it is reasonable to suppose that in estimating spectral counts, as with many other real-world values, it is the percentage error, not the absolute error that matters. Thus, the appropriate noise model would appear to be multiplicative, not additive. Logarithms conveniently convert multiplicative noise to additive noise, thus

Table 4.1: Spearman rank correlation coefficients on the Mouse Brain dataset

Models	22 features	42 features	62 features	232 features
LIN1	0.1826	0.2424	0.2135	0.2538
LIN2	0.2326	0.2520	0.2300	0.1996
LIN3	0.4433	0.4426	0.4492	0.4532
INV1	0.3107	0.2809	0.2337	0.1965
INV2	0.4968	0.4932	0.3867	0.2508
INV3	0.4971	0.4942	0.5030	0.5.55
EXP1	0.5134	0.5118	0.5222	0.4171
EXP2	0.5184	0.5178	0.1224	0.5435
LIND	0.4087	0.4087	0.4087	0.4087
INVD	0.4215	0.4215	0.4215	0.4215
EXPD	0.4664	0.4664	0.4664	0.4664
LINR	0.3783	0.3783	0.3783	0.3783
INVR	0.3867	0.3867	0.3867	0.3867
EXPR	0.4566	0.4566	0.4566	0.4566

making linear regression a natural method to apply to the transformed data, which is exactly what the two exponential methods do. One thing to note is the bad performance of EXP2 model when using 62 features, from our experiments, this is because of the not full column rank of the matrix V^TV mentioned in Section 4.2.3. Therefore, before applying EXP2, a pre-check of full-rankness is a necessity. In addition, unlike the inverse transformation, logarithms map large values to large values, and small values to small values. In contrast, since the inverse transformation maps large values to small values, a small error in the estimated value of $1/y$ can result in a large error in y when y is large. This may more than compensate for the ability of the inverse model to eliminate all skew

Table 4.2: Spearman rank correlation coefficients on the Mouse Heart dataset

Models	22 features	42 features	62 features	232 features
LIN1	0.1792	0.1699	0.1559	0.1290
LIN2	0.1599	0.1629	0.1681	0.1674
LIN3	0.4939	0.4926	0.4002	0.4303
INV1	0.3186	0.2525	0.2366	0.2634
INV2	0.5385	0.5328	0.3511	0.2092
INV3	0.5322	0.5274	0.5310	0.4963
EXP1	0.5495	0.5463	0.5531	0.4963
EXP2	0.5511	0.5266	0.1635	0.5684
LIND	0.4685	0.4685	0.4685	0.4685
INVD	0.5132	0.5132	0.5132	0.5132
EXPD	0.5137	0.5137	0.5137	0.5137
LINR	0.4436	0.4436	0.4436	0.4436
INVR	0.4523	0.4523	0.4523	0.4523
EXPR	0.5033	0.5033	0.5033	0.5033

from the data.

However, even the exponential methods perform only slightly better than the best naive method (which, as it turns out, is the exponential naive method, EXPD). The naive methods essentially estimate the protein level by taking an average of the peptide spectral counts for that protein. That they perform as well as they do can be interpreted in a straightforward way. Higher protein levels will certainly give rise to higher peptide spectral count. Apparently, the reverse is also true to an extent: higher spectral counts are, on average, a reflection of higher protein levels. Put another way, two factors contribute to spectral counts: the input level of the protein, and the ionization efficiency of

Table 4.3: Spearman rank correlation coefficients on the Mouse Kidney dataset

Models	22 features	42 features	62 features	232 features
LIN1	0.1540	0.2186	0.1826	0.2299
LIN2	0.2112	0.1857	0.1685	0.1884
LIN3	0.4678	0.4685	0.4692	0.2027
INV1	0.3191	0.2784	0.2238	0.1826
INV2	0.5130	0.5049	0.3198	0.2299
INV3	0.5117	0.5098	0.5101	0.2314
EXP1	0.5214	0.5091	0.5240	0.3894
EXP2	0.5254	0.5266	0.1855	0.5338
LIND	0.4215	0.4215	0.4215	0.4215
INVD	0.4215	0.4215	0.4215	0.4215
EXPD	0.4695	0.4695	0.4695	0.4695
LINR	0.3947	0.3947	0.3947	0.3947
INVR	0.3998	0.3998	0.3998	0.3998
EXPR	0.4584	0.4584	0.4584	0.4584

the peptide. The naive methods focus entirely on the former factor, which by itself is enough to garner a correlation coefficient of about 0.5. However, the naive methods do nothing to explain the very large differences in spectral count between different peptides from the same protein. Instead, they assume the spectral counts are all of equal intensity, something that is not observed experimentally.

Finally, we should point out that of the four different vector representations, the 22-feature vector generally works the best, though the 42-feature and 62-feature vectors are often competitive. This suggests that sequential information (as captured in these latter two vectors) is not as important as good statistics on amino-acid counts (as captured

in the 22-feature vector). The generally poor performance of the 232-feature vector suggests that it is overfitting, since it includes all the features of the 22-feature vector, which performs best. However, theses comments apply only when we consider all of the fitting methods. When we consider only EXP1 and EXP2, the best performing methods, their performance seems independent of which vector representation is used.

4.4.3 Visualization of the Goodness of Fit

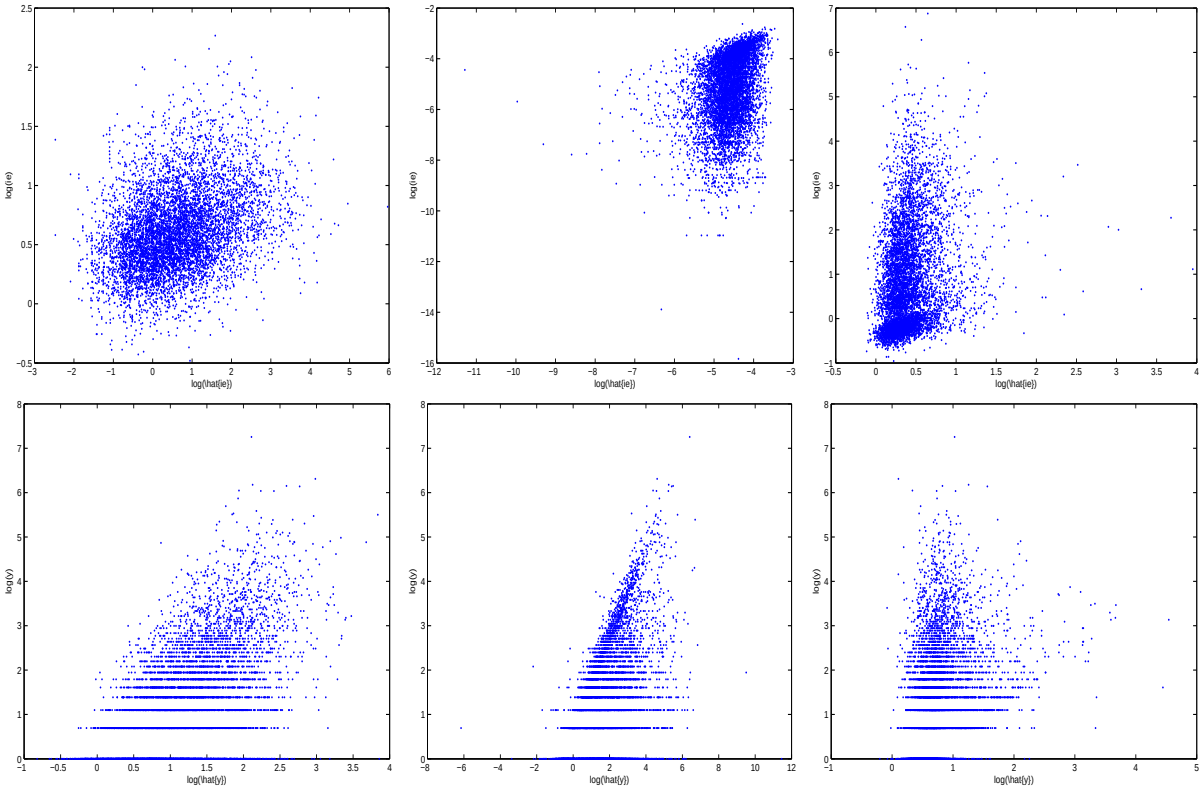

Figure 4.7: Log-log plots of y v.s. $\hat{y}$ (bottom row) and ie v.s. $\hat{ie}$ (top row) on the Kidney dataset. The two left panels are the results of EXP1 method, the two middle panels are the results of the LIN3 method, and the two right panels are the results of the INV3 method.

As mentioned above, the three methods LIN3, INV3 and EXP1 comes from different families: linear models, inverse models, exponential models, respectively. To help visu-

alize the goodness of the fit provided by these methods, Figure 4.7 provides log-log plots of observed and estimated values for a training dataset.[5] For each method, we provide two figures, a plot of y v.s. $\hat{y}$, and a plot of ie v.s. $\hat{ie}$. Here, ie and $\hat{ie}$ are derived from y and $\hat{y}$, respectively, by dividing by $\hat{in}$, the estimate of protein input level. ie and $\hat{ie}$ thus provide two different measures of the ionization efficiency of a peptide. Note that $\hat{ie}$ is the same as the estimate described in Section 4.3.

The first thing to notice is the strong horizontal lines in the plots of y v.s. $\hat{y}$. These are due to the discrete nature of the observed values, y, most of which take on small, positive integer values. The plots have no such vertical lines because the estimates, $\hat{y}$, are real valued. The second thing to notice is that the plots of ie v.s. $\hat{ie}$ have no strong lines. This is because ionization efficiency is inherently real-valued. Formally, dividing y by $\hat{in}$ produces a real number, since $\hat{ie}$ is real valued. Finally, notice that the plot of ie v.s. $\hat{ie}$ for the EXP1 method is by far the most Gaussian-looking of all the plots. This is another measure by which it provides the best fit to the experimental data, in addition to its relatively high correlation coefficient. In contrast, the plots for LIN3 and INV3 all show evidence of residual structure that the methods were unable to fit.

4.5 Conclusions of Regression Analysis

We develop and evaluate a number of methods for estimating protein levels and peptide spectral counts in Tandem Mass Spectrometry (MS/MS) data. Other researchers have attempted to estimate the peptide spectral counts in an MS/MS spectrum given the amount of protein input to a mass spectrometer [21]. However, in this chapter, we addressed the reverse problem of estimating the amount of protein input given an MS/MS spectrum. A solution to this problem would allow biologists to efficiently estimate the amounts of the thousands of proteins in a tissue sample. To our knowledge, this is the

[5]The INV3 and LIN3 methods produce a very small number of negative estimates, which we discard before taking logs.

first attempt to solve this problem using large amounts of data.

The methods we developed are based on simple, generative models of MS/MS data. In fitting the models to the data, the methods attempt to solve two problems simultaneously: estimating protein input levels, and explaining why different peptides produce peaks of different intensity. Of the eight methods developed, the two exponential methods, EXP1 ad EXP2, performed the best. However, this research is just a first step, and additional work is needed before protein levels and spectral counts can be predicted accurately.

The results here suggest several directions for future work:

(*i*)More sophisticated regression methods, such as kernel methods and regression trees could be tried. String kernels, in particular, could extract more information from the peptide sequences. Such methods could easily be adapted to the framework of our EXP1 method, which transforms the data into a more manageable form while simultaneously factoring out the unknown amount of protein input.

(*ii*) The problem of a fit being dominated by a few extremely large data values might be ameliorated by a weighting scheme that places a large weight on small values.

(*iii*)Although we tested several representations of peptides in this paper, it is not yet clear whether the solution lies in more sophisticated regression schemes or in better representations of peptides. Representations that capture more sequential information (such as string kernels) may be important. In addition, a more refined representation of peptide spectral counts may be needed. For instance,spectral peak shape, not just values of spectral counts, may be important. To this end, we have recently acquired more detailed MS/MS data that allows us to estimate peak shape.

(*iv*)The possible problem of overfitting can be addressed by regularization and bagging. In particular, regularized linear regression could easily be incorporated into our EXP1 method in order to test the utility of large vector representations of peptides.

Table 4.4: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Brain dataset with 22 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.1823	0.0064	0.1826	0.0384
LIN2	0.2262	0.0070	0.2326	0.0511
LIN3	0.4017	0.0054	0.4433	0.0550
INV1	0.3164	0.0046	0.3107	0.0377
INV2	0.5030	0.0047	0.4968	0.0525
INV3	0.3954	0.0056	0.4971	0.0495
EXP1	0.5210	0.0046	0.5134	0.0455
EXP2	0.5249	0.0047	0.5184	0.0432
LIND	0.4100	0.0059	0.4087	0.0563
INVD	0.4647	0.0058	0.4215	0.0356
EXPD	0.4692	0.0055	0.4664	0.0551
LINR	0.3839	0.0062	0.3783	0.0485
INVR	0.3915	0.0056	0.3867	0.0516
EXPR	0.4586	0.0051	0.4566	0.0497

Table 4.5: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Brain dataset with 22 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.3054	0.0115	0.3008	0.0483
LIN2	0.3088	0.0190	0.2943	0.0772
LIN3	0.3966	0.0051	0.3598	0.0365
INV1	0.2834	0.0051	0.2834	0.0402
INV2	0.2490	0.0081	0.2412	0.0298
INV3	0.2872	0.0064	0.2479	0.0368
EXP1	0.3432	0.0044	0.3312	0.0358
EXP2	0.3489	0.0044	0.3379	0.0326
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0746	0.0144	0.0916	0.0654
INVR	0.0682	0.0146	0.0641	0.0400
EXPR	0.0640	0.0131	0.0513	0.0379

Table 4.6: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Brain dataset with 42 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2215	0.0088	0.2424	0.0610
LIN2	0.2416	0.0190	0.2520	0.0423
LIN3	0.4039	0.0057	0.4426	0.0565
INV1	0.2798	0.0045	0.2809	0.0447
INV2	0.5031	0.0048	0.4932	0.0531
INV3	0.3967	0.0061	0.4942	0.0511
EXP1	0.5235	0.0048	0.5118	0.0508
EXP2	0.5286	0.0049	0.5178	0.0470
LIND	0.4100	0.0059	0.4087	0.0563
INVD	0.4647	0.0058	0.4215	0.0356
EXPD	0.4692	0.0055	0.4664	0.0551
LINR	0.3839	0.0062	0.3783	0.0485
INVR	0.3915	0.0056	0.3867	0.0516
EXPR	0.4586	0.0051	0.4566	0.0497

Table 4.7: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Brain dataset with 42 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2874	0.0043	0.2860	0.0442
LIN2	0.2987	0.0154	0.2885	0.0659
LIN3	0.4077	0.0049	0.3631	0.0356
INV1	0.2454	0.0056	0.2421	0.0329
INV2	0.2504	0.0082	0.2341	0.0364
INV3	0.2977	0.0062	0.2481	0.0373
EXP1	0.3503	0.0039	0.3324	0.0405
EXP2	0.3596	0.0042	0.3421	0.0350
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0746	0.0144	0.0916	0.0654
INVR	0.0682	0.0146	0.0641	0.0400
EXPR	0.0640	0.0131	0.0513	0.0379

Table 4.8: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Brain dataset with 62 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2102	0.0098	0.2135	0.0573
LIN2	0.2290	0.0138	0.2300	0.0537
LIN3	0.4138	0.0059	0.4492	0.0582
INV1	0.2501	0.0175	0.2337	0.0390
INV2	0.3769	0.1435	0.3867	0.1512
INV3	0.4152	0.0054	0.5030	0.0480
EXP1	0.5377	0.0046	0.5222	0.0497
EXP2	0.1155	0.0729	0.1224	0.0725
LIND	0.4100	0.0059	0.4087	0.0563
INVD	0.4647	0.0058	0.4215	0.0356
EXPD	0.4692	0.0055	0.4664	0.0551
LINR	0.3839	0.0062	0.3783	0.0485
INVR	0.3915	0.0056	0.3867	0.0516
EXPR	0.4586	0.0051	0.4566	0.0497

Table 4.9: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Brain dataset with 62 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.3143	0.0132	0.3216	0.0440
LIN2	0.2928	0.0183	0.2989	0.0598
LIN3	0.4264	0.0049	0.3784	0.0358
INV1	0.2681	0.0284	0.2795	0.0599
INV2	0.2659	0.0482	0.2551	0.0625
INV3	0.3306	0.0056	0.2748	0.0370
EXP1	0.3881	0.0044	0.3630	0.0407
EXP2	0.0712	0.0515	0.4376	0.0838
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0746	0.0144	0.0916	0.0654
INVR	0.0682	0.0146	0.0641	0.0400
EXPR	0.0640	0.0131	0.0513	0.0379

Table 4.10: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Brain dataset with 232 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2510	0.0402	0.2538	0.0523
LIN2	0.1981	0.0135	0.1996	0.0382
LIN3	0.4434	0.0057	0.4532	0.0505
INV1	0.1986	0.0133	0.1965	0.0647
INV2	0.2565	0.0861	0.2508	0.0690
INV3	0.4704	0.0070	0.5055	0.0524
EXP1	0.4275	0.1495	0.4171	0.1231
EXP2	0.5864	0.0050	0.5435	0.0416
LIND	0.4100	0.0059	0.4087	0.0563
INVD	0.4647	0.0058	0.4215	0.0356
EXPD	0.4692	0.0055	0.4664	0.0551
LINR	0.3839	0.0062	0.3783	0.0485
INVR	0.3915	0.0056	0.3867	0.0516
EXPR	0.4586	0.0051	0.4566	0.0497

Table 4.11: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Brain dataset with 232 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2630	0.0547	0.2526	0.0766
LIN2	0.2946	0.0126	0.2953	0.0448
LIN3	0.4974	0.0062	0.4075	0.0337
INV1	0.2091	0.0807	0.2121	0.0992
INV2	0.2688	0.0340	0.2710	0.0549
INV3	0.4157	0.0064	0.3387	0.0334
EXP1	0.4379	0.0456	0.4301	0.0719
EXP2	0.4959	0.0060	0.4397	0.0320
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0746	0.0144	0.0916	0.0654
INVR	0.0682	0.0146	0.0641	0.0400
EXPR	0.0640	0.0131	0.0513	0.0379

Chapter 5

Canonical Correlation Analysis

5.1 Introduction

[1]In this chapter, we investigate the possibility of predicting protein abundance with an existing generative model, called Canonical Correlation Analysis (CCA) [25], which can be viewed as an extension of multiple linear regression. Although it is a standard tool of multivariate statistical analysis and has been used, for example, in economics, medicine, meteorology and even in the classification of malt whiskey [31], CCA is surprisingly unknown in the field of data mining, some exceptions being [27, 17, 2]. Because this is an initial study, we chose it for its simplicity and tractability, and the goal was to see how well (or poorly) it fits the data, and to quantify the error. The model predicts the spectral count of a peptide based on two factors: its amino-acid sequence, and the abundance of the protein from which it was derived. The model provides an explanation for the linear relationship between protein abundance and spectral count. More importantly, we show how to use the model to estimate protein abundance from spectral count.

However, while we have found that CCA is quite adequate for small data sets, we have also found that it is computationally expensive for the large data sets we are dealing

[1]this chapter also appeared in [7]

with. As an alternative, this chapter explores that LIN3 model developed in Chapter 4 can be viewed as an approximation of CCA, one that avoids the need to deal with large, dense matrices. LIN3 is correct for data of a certain form, which includes the kind of data we are dealing with. We trained and tested both CCA and LIN3 on the biological datasets mentioned before.The main evaluation method was 10-fold cross validation, with correlation coefficient used to measure the goodness-of-fit of a learned model to the testing portion of the data. The main difficulty in carrying out the evaluation was the distribution of the spectral counts, which ranges over several orders of magnitude and is highly skewed, with most data concentrated at very low values. To deal with this difficulty, we use the same strategy as before, ie. we use the Spearman rank correlation coefficient to measure the goodness-of-fit.

The rest of this chapter is organized as follows. Section 5.2 reviews the mathematical background of CCA, and how to use the CCA model to fit mass spectromety data. Section 5.2.3 discusses the computational bottlenecks of CCA and show why LIN3 is an efficient and robust approximation. Section 5.3 discusses generalization and regularization issues. Section 5.4 presents our experimental results. Finally, Section 5.5 presents conclusions and possible extensions.

5.2 Canonical Correlation Analysis (CCA)

5.2.1 Mathematical Background

Canonical correlation analysis (CCA), first developed by H. Hotelling [25], is a way of measuring the linear relationship between two multidimensional variables, $\mathbf{X}$ and $\mathbf{Y}$. This can be defined as the problem of finding two sets of basis vectors, one for $\mathbf{X}$ and the other for $\mathbf{Y}$, such that the correlations between the projections of the variables onto these basis vectors are mutually maximized. Consider the linear combinations $x = \mathbf{X}^T\mathbf{w}_x$ and $y = \mathbf{Y}^T\mathbf{w}_y$ of the two variables respectively. This means that the function to be

maximized is

$$\begin{aligned}\rho &= \frac{E[x \cdot y]}{\sqrt{E[x^2]E[y^2]}} = \frac{E[\mathbf{w}_x^T \mathbf{X}\mathbf{Y}^T \mathbf{w}_y]}{\sqrt{E[\mathbf{w}_x^T \mathbf{X}\mathbf{X}^T \mathbf{w}_x]E[\mathbf{w}_y^T \mathbf{Y}\mathbf{Y}^T \mathbf{w}_y]}} \\ &= \frac{\mathbf{w}_x^T \mathbf{C}_{xy} \mathbf{w}_y}{\sqrt{\mathbf{w}_x^T \mathbf{C}_{xx} \mathbf{w}_x \mathbf{w}_y^T \mathbf{C}_{yy} \mathbf{w}_y}}\end{aligned} \tag{5.1}$$

Here, $\mathbf{C}_{xy}$ is the covariance matrix of $\mathbf{X}$ and $\mathbf{Y}$, and $\mathbf{w}_x$ and $\mathbf{w}_y$ are corresponding matrices, the columns of which form the pairwise orthogonal basis vectors. The subsequent canonical correlations are uncorrelated for different solutions. That is, when $i \neq j$,

$$\begin{cases} E[x_i x_j] = E[\mathbf{w}_{xi}^T \mathbf{X}\mathbf{X}^T \mathbf{w}_{xj}] = \mathbf{w}_{xi}^T \mathbf{C}_{xx} \mathbf{w}_{xj} = 0 \\ E[y_i y_j] = E[\mathbf{w}_{yi}^T \mathbf{Y}\mathbf{Y}^T \mathbf{w}_{yj}] = \mathbf{w}_{yi}^T \mathbf{C}_{yy} \mathbf{w}_{yj} = 0 \\ E[x_i y_j] = E[\mathbf{w}_{xi}^T \mathbf{X}\mathbf{Y}^T \mathbf{w}_{yj}] = \mathbf{w}_{xi}^T \mathbf{C}_{xy} \mathbf{w}_{yj} = 0 \end{cases} \tag{5.2}$$

The random variables x and y (*i.e.*, the projections of $\mathbf{X}$ and $\mathbf{Y}$ onto $\mathbf{w}_x$ and $\mathbf{w}_y$, respectively) are called *canonical variables*.

To find the canonical correlations between the random, multidimensional variables, $\mathbf{X}$ and $\mathbf{Y}$, assume without loss of generality that the variables have mean 0. Their total covariance matrix is therefore

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{xx} & \mathbf{C}_{xy} \\ \mathbf{C}_{yx} & \mathbf{C}_{yy} \end{bmatrix} = E\left[\begin{pmatrix} \mathbf{X} \\ \mathbf{Y} \end{pmatrix} \begin{pmatrix} \mathbf{X} \\ \mathbf{Y} \end{pmatrix}^T \right] \tag{5.3}$$

This is a block matrix where $\mathbf{C}_{xx}$ and $\mathbf{C}_{yy}$ are the within-sets covariance matrices of $\mathbf{X}$ and $\mathbf{Y}$, respectively, and $\mathbf{C}_{xy} = \mathbf{C}_{yx}^T$ is the between-sets covariance matrix. The canonical correlations can then be found by solving the following generalized eigenvalue equations:

$$\begin{cases} \rho^2 \mathbf{C}_{xx} \beta = \mathbf{C}_{xy} \mathbf{C}_{yy}^{-1} \mathbf{C}_{yx} \beta \\ \rho^2 \mathbf{C}_{yy} \alpha = \mathbf{C}_{yx} \mathbf{C}_{xx}^{-1} \mathbf{C}_{xy} \alpha \end{cases} \tag{5.4}$$

Here, the eigenvalues ρ^2 are the squared *canonical correlations*, and the eigenvectors β and α are the normalized canonical correlation *basis vectors*. The number of non-zero solutions to these equations is limited to the smallest dimensionality of $\mathbf{X}$ and $\mathbf{Y}$.

In this chapter, we are interested in the eigenvectors that corresponds to the largest eigenvalue; that is, we want the largest canonical correlation. Thus, we want to find two

parameter vectors α and β such that the correlation coefficient between the two variables $x = \mathbf{X}^T\beta$ and $y = \mathbf{Y}^T\alpha$ is maximal.

5.2.2 Applying CCA in Mining MS/MS data

Our goal is to fit a parametric model to the MS/MS data. As shown in a parallel paper [8], this problem can be modeled under a linear regression framework. The model uses the following variables: P_i is a protein; p_{i1}, p_{i2},...,p_{ik_i} are the peptides fragmented from the protein in an MS/MS experiment; $y_{i1}, y_{i2}, ...y_{ik_i}$ are the spectral counts of these peptides in the protein's mass spectrum; in_i is the amount of protein P_i in the protein mixture that is being analyzed by the MS/MS experiment. Note that in_i is a latent variable whose value we do not know. The following equation provides a simple model of the generation of spectral peaks:

$$y_{ij} \;=\; in_i \cdot ie_{ij} + \varepsilon_{ij} \tag{5.5}$$

This equation divides the peak intensity into two factors: in_i, the amount of protein from which the peptide was generated; and ie_{ij}, the *ionization efficiency* of peptide p_{ij}. Ionization efficiency can be thought of as the propensity of the peptide to ionize and contribute to a peak, though it includes all factors that contribute to peak intensity *other than* the amount of protein. In this way, we hope to untangle the amount of protein (which we want to estimate) from all other factors. ε_{ij} is simply the error in this model.

To estimate the ionization efficiency of a peptide, we model it as a linear function with error:

$$ie_{ij} \;=\; x_{ij} \bullet \beta + \varepsilon'_{ij} \tag{5.6}$$

Here, β is a vector of parameters (to be learned), x_{ij} is a vector of peptide properties (derived from the peptide sequence), and $\bullet$ denotes the dot product (or inner product) of the two vectors. The peptide properties could include such things as length, mass, amino-acid composition, and estimates of various biochemical properties such as hydrophobicity, chargeablity, etc. ε'_{ij} is the error in the linear model.

Combining the two equations above gives our overall model:

$$y_{ij} \;=\; in_i \cdot (x_{ij} \bullet \beta) + \varepsilon''_{ij} \tag{5.7}$$

Observe that the right-hand side contains a product of two unknowns, in_i and β. To eliminate this non-linear term, we divide both sides by in_i, to obtain a linear model:

$$y_{ij} \cdot \alpha_i \;=\; x_{ij} \bullet \beta + \varepsilon'''_{ij} \tag{5.8}$$

where $\alpha_i = 1/in_i$. Additional details of this model can be found in [6]. We call this an "extended" regression model because of the presence of unknown values on both sides of the equation. Such models cannot be solved by the classical techniques of linear regression. However, we use CCA to provide an elegant solution.

First, we construct a sparse matrix, $\mathbf{Y}$, as follows:

$$\mathbf{Y} = \left(\begin{array}{ccc|ccc|c|ccc} y_{11} & \cdots & y_{1k_1} & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & y_{21} & \cdots & y_{2k_2} & \cdots & \ddots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & & & \ddots & \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 & & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & y_{l_l} & \cdots & y_{lk_l} \end{array}\right)$$

where l is the number of proteins. In this matrix, each row corresponds to a protein, and each column to a peptide. Element y_{ij} is the value of the spectral count of the j^{th} peptide of protein P_i.

We also construct the following matrix:

$$\mathbf{X} \;=\; (x_{11}, ..., x_{1k_1}, x_{21}, ..., x_{2k_2}, ..., x_{l1}, ..., x_{lk_l})$$

Here, each x_{ij} is a column vector, the vector of properties of peptide p_{ij}, defined above.

In addition to these two (known) matrices, $\mathbf{X}$ and $\mathbf{Y}$, we define two (unknown) column vectors, α and β. β was defined above, and α is defined to be $(\alpha_1, ..., \alpha_l)^T$, where $\alpha_i = 1/in_i$ as defined above. Of course, our aim is to estimate values for α and β.

With these definitions, Equation 5.8 can be rewritten as:

$$\mathbf{Y}^T\alpha \;=\; \mathbf{X}^T\beta + E \tag{5.9}$$

where E is a column vector of errors. We use CCA to find values for α and β that maximize the correlation coefficient between the two random vectors $\mathbf{Y}^T\alpha$ and $\mathbf{X}^T\beta$. From Equation 5.4, the solution to α and β can be found by solving the following system of generalized eigenvector equations:

$$\begin{cases} \rho^2(\mathbf{X}\mathbf{X}^T)\hat{\beta} = \mathbf{X}\mathbf{Y}^T(\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{Y}\mathbf{X}^T\hat{\beta} \\ \rho^2(\mathbf{Y}\mathbf{Y}^T)\hat{\beta} = \mathbf{Y}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{Y}^T\hat{\beta} \end{cases} \tag{5.10}$$

Because the eigenvalue, ρ^2, represents the squared correlation coefficient between the two random vectors $\mathbf{Y}^T\alpha$ and $\mathbf{X}^T\beta$, we choose the eigenvectors, α and β, with the largest eigenvalue.

Although elegant, we have found that solving the two equations 5.10 is computationally intractable. The main problem is the large size of the matrix $\mathbf{Y}$. The data we are dealing with contains about 10,000 different peptides and 2,000 different proteins. Matrix $\mathbf{Y}$ therefore has dimensions $2,000 \times 10,000$, and so the matrix $\mathbf{Y}\mathbf{Y}^T$ has dimensions $2,000 \times 2,000$. The first equation in 5.10 requires inverting this matrix. However, inverting such a large matrix requires an inordinate amount of time and space, and leads to severe numerical problems [11]. The second Equation in 5.10 requires inverting the matrix $\mathbf{X}\mathbf{X}^T$, which is not nearly so large and can easily be inverted. However, on the left side of the equation, we again encounter the large matrix $\mathbf{Y}\mathbf{Y}^T$, which makes the generalized eigenvector decomposition extremely expensive.

Because of these problems, we must try to find an approximation to CCA, one that is much more efficient and robust, as shown in the next section.

5.2.3 The Relationship between CCA and LIN3

In this section, we will show that LIN3 is in fact a more efficient and robust approximation of the CCA model under some specific settings. Although it is very large, the matrix $\mathbf{Y}$ defined in the previous section is also very sparse. In every column, only one element is non-zero. By exploiting the structure and sparseness of this matrix, an efficient approximation of CCA could be developed.

In CCA, the statistical measure of similarity between two random vectors is *correlation coefficient*. Formally, we want to find α and β that maximize the following expression:

$$\frac{(\mathbf{Y}^T\alpha - \overline{\mathbf{Y}^T\alpha}) \bullet (\mathbf{X}^T\beta - \overline{\mathbf{X}^T\beta})}{\|\mathbf{Y}^T\alpha - \overline{\mathbf{Y}^T\alpha}\| \cdot \|\mathbf{X}^T\beta - \overline{\mathbf{X}^T\beta}\|} \tag{5.11}$$

The point to notice here is that the two vectors are centered, by subtracting their means. The correlation coefficient is thus the cosine of the angle between the two centered vectors, and CCA finds α and β to minimize this angle. Unfortunately, from a computational point of view, the centering of the vectors causes a great deal of problems, because it effectively destroys the sparse structure of matrix $\mathbf{Y}$. This is because the large covariance matrix, $\mathbf{C}_{yy}$, is defined not in terms of $\mathbf{Y}$, but in terms of $\mathbf{Y} - \overline{\mathbf{Y}}$. Unfortunately, although $\mathbf{Y}$ is sparse, $\overline{\mathbf{Y}}$ is dense, so $\mathbf{Y} - \overline{\mathbf{Y}}$ is also dense. In fact, in row i of matrix $\overline{\mathbf{Y}}$, each entry is $\sum_{j=1}^{n_i} y_{ij}/N$, so $\overline{\mathbf{Y}}$ has no zeros and is maximally dense. Likewise for $\mathbf{Y} - \overline{\mathbf{Y}}$.

In the approximation method developed here, we retain the idea of minimizing the angle, but without the requirement of centering the vectors first. That is, we minimize the angle between the uncentered vectors $\mathbf{Y}^T\alpha$ and $\mathbf{X}^T\beta$, by maximizing the cosine of the angle between them. This amounts to maximizing the following expression:

$$\frac{\mathbf{Y}^T\alpha \bullet \mathbf{X}^T\beta}{\|\mathbf{Y}^T\alpha\| \cdot \|\mathbf{X}^T\beta\|} \tag{5.12}$$

Recall the our discussion in Chapter 4, it's easy to see that this is exactly the model LIN3. Maximizing this expression leads to the same generalized eigenvector equations given in

Equation 5.4, except that now the covariance matrices are defined in terms of uncentered random variables. Thus, instead of $\mathbf{C}_{yy} = (\mathbf{Y} - \overline{\mathbf{Y}})(\mathbf{Y} - \overline{\mathbf{Y}})^T$, we now use $\mathbf{C}_{yy} = \mathbf{Y}\mathbf{Y}^T$.

Of course, this does not change the dimensions of any of the covariance matrices. In particular, $\mathbf{C}_{yy}$ is still very large. However, it is now possible to simplify Equations 5.4 so that the remaining matrices are relatively small. This is shown in Theorem 1 below. In this theorem, Y_i is the column vector $(y_{i1}, y_{i2}, ..., y_{in_i})^T$, and $\mathbf{X}_i$ is the matrix $(x_{i1}, x_{i2}, ..., x_{in_i})$. They represent, respectively, the spectral peak intensities and peptide properties for protein i. Note that $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_N)$, so each $\mathbf{X}_i$ is a vertical slice of the larger matrix $\mathbf{X}$.

Theorem 1: *Expression 5.12 above is maximized when the parameter vector β is a solution of the following generalized eigenvector equation:*

$$\rho^2 \mathbf{X}\mathbf{X}^T \beta \;=\; [\sum_i \mathbf{X}_i Y_i Y_i^T \mathbf{X}_i^T / \|Y_i\|^2]\, \beta \tag{5.13}$$

Moreover, it is the eigenvector with the largest eigenvalue, ρ^2. In addition, $\rho = \cos\theta$, where θ is the angle between the vectors $\mathbf{Y}^T\alpha$ and $\mathbf{X}^T\beta$. Finally, the elements of the parameter vector α are given by the following equation:

$$\alpha_i \;=\; \mathbf{Y}_i^T \mathbf{X}_i^T \beta / \rho \|Y_i\|^2 \tag{5.14}$$

Proof: First, observe that the maximum of expression 5.12 is the same as the maximum of the simpler expression $\mathbf{A}Y \bullet \mathbf{X}^T\beta$ subject to the constraints $\|\mathbf{A}Y\| = 1$ and $\|\mathbf{X}^T\beta\| = 1$. To carry out this constrained maximization, we use Lagrange multipliers and maximize the following expression:

$$\mathbf{A}Y \bullet \mathbf{X}^T\beta \;-\; \lambda(\|\mathbf{A}Y\|^2 - 1) \;-\; \mu(\|\mathbf{X}^T\beta\|^2 - 1)$$

It is not hard to see that this expression is equivalent to the following:

$$\sum_i \alpha_i Y_i^T \mathbf{X}_i^T \beta \;-\; \lambda(\sum_i \|\alpha_i Y_i\|^2 - 1) \;-\; \mu(\|\mathbf{X}^T\beta\|^2 - 1)$$

Taking partial derivatives with respect to β and α_i and setting the results to 0 gives the following equations:

$$\sum_i \alpha_i \mathbf{X}_i Y_i = 2\mu \mathbf{X}\mathbf{X}^T \beta \quad (5.15)$$

$$Y_i^T \mathbf{X}_i^T \beta = 2\lambda \alpha_i \|Y_i\|^2 \quad (5.16)$$

Left-multiplying Equation 5.15 by β^T gives

$$\beta^T \sum_i \alpha_i \mathbf{X}_i Y_i = 2\mu \beta^T \mathbf{X}\mathbf{X}^T \beta \quad (5.17)$$

$$= 2\mu \|\mathbf{X}^T \beta\|^2 \quad (5.18)$$

$$= 2\mu \quad (5.19)$$

since, by our constraint, $\|\mathbf{X}^T \beta\| = 1$. In a similar fashion, multiplying Equation 5.16 by α_i and summing over i gives

$$\sum_i \alpha_i Y_i^T \mathbf{X}_i^T \beta = 2\lambda \sum_i \alpha_i^2 \|Y_i\|^2 \quad (5.20)$$

$$= 2\lambda \|\mathbf{A}Y\|^2 \quad (5.21)$$

$$= 2\lambda \quad (5.22)$$

since, by our constraint, $\|\mathbf{A}Y\| = 1$. Note that Equations 5.19 and 5.22 can be rewritten as follows:

$$2\lambda = \mathbf{A}Y \bullet \mathbf{X}^T \beta = 2\mu \quad (5.23)$$

In other words, $\lambda = \mu = \rho/2$, where ρ is the value we are maximizing. From this and Equation 5.16, it follows immediately that

$$\alpha_i = Y_i^T \mathbf{X}_i^T \beta / \rho \|Y_i\|^2 \quad (5.24)$$

This proves Equation 5.14. To prove Equation 5.13, note that from Equations 5.15 and 5.24, we get

$$2\mu\rho \mathbf{X}\mathbf{X}^T \beta = [\sum_i \mathbf{X}_i Y_i Y_i^T \mathbf{X}_i^T / \|Y_i\|^2] \beta$$

The result follows immediately, since $2\mu\rho = \rho^2$.

End of Proof.

Comparing Equation 5.13 in this theorem with Equation 5.4 in Section 5.2, the important point is that we no longer need to compute or invert the large covariance matrix $\mathbf{C}_{yy}$. As for the other matrices, $\mathbf{X}_i$ has dimensions $p \times n_i$, where p is the number of properties (or features) characterizing each peptide. Matrix $\mathbf{X}$ thus has dimensions $p \times M$, where $M = \sum_i n_i$ is the total number of peptides. This means that the matrices $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}_i Y_i Y_i^T \mathbf{X}_i^T$ have dimensions $p \times p$. It is the size of these $p \times p$ matrices that is important, since this determines the cost of solving eigenvector equation 5.13.

Since our data sets contain about 10,000 peptides, M is large. However, p, the number of peptide features is much smaller. In this paper, we use two different feature sets, one with $p = 21$ and one with $p = 421$, as described in Section 5.4. With $p = 21$ the eigenvector matrices are very small. With $p = 421$ they are much larger, and although they slowed down the eigenvector computations discernably, they still posed no significant problems. Moreover, they are considerably smaller than the covariance matrix $\mathbf{C}_{yy}$ in Equation 5.4, which has dimensions 2000×2000 and did cause significant computational problems. Perhaps more importantly, the size of the $p \times p$ matrices is independent of how many proteins or peptides are in the data set. Thus, computational cost will not be significantly affected as the data sets grow.

5.3 Generalization and Regularization Issues

In fitting a model to data, one may not wish to treat all data points equally, but to place different importance on different data. To allow for this, one can introduce weights into the optimization criterion. For LIN3, the optimization criterion is given by Equation 5.12, which specifies the angle between two vectors. In this case, we can define what might be called a generalized angle, in which different vector components are weighted differently.

If the two vectors are denoted U and V, then the generalized angle is defined by (the arc cosine of) the following expression:

$$\frac{U^T \mathbf{W} V}{\sqrt{U^T \mathbf{W} U}\,\sqrt{V^T \mathbf{W} V}}$$

Here, $\mathbf{W}$ is a diagonal matrix, whose i^{th} diagonal element, $\mathbf{w}_i$, is the weight of the i^{th} component of the vectors. When $\mathbf{W}$ is the identity matrix, we get the ordinary, unweighted angle. To compute the generalized angle between U and V, we can use the above formula, or we can first transform the vectors as follows:

$$U_i' \;=\; U_i\sqrt{\mathbf{w}_i} \qquad\qquad V_i' \;=\; V_i\sqrt{\mathbf{w}_i}$$

and then compute the unweighted angle between U' and V'. This latter approach shows that giving weight $\mathbf{w}_i$ to data point (U_i, V_i) is equivalent to simply multiplying U_i and V_i by $\sqrt{\mathbf{w}_i}$. This makes intuitive sense, since the angle between two vectors is more strongly influenced by large vector components than by small ones.

In the case of our MS/MS data, this corresponds to assigning a different weight to each peptide, and to transforming each peptide's peak intensity and feature vector as follows:

$$y_{ij}' \;=\; y_{ij}\sqrt{\mathbf{w}_{ij}} \qquad\qquad x_{ij}' \;=\; x_{ij}\sqrt{\mathbf{w}_{ij}}$$

where $\mathbf{w}_{ij}$ is the weight assigned to peptide j of protein i. We then apply Theorem 1 to y_{ij}' and x_{ij}', instead of to y_{ij} and x_{ij}. The choice of what weights to use is heuristic, and in our experiments, we chose two different sets of weights, $\mathbf{w}_{ij} = ||Y_i||$ and $\mathbf{w}_{ij} = 1/||Y_i||$, respectively. These weights are a simple attempt to address two different sources of noise and error. The first set of weights emphasizes peptides from proteins with large spectral counts, since they have a higher resolution and a better signal-to-noise ratio. The second set of weights attempts to stabilize the model error, assuming that peptides from proteins with larger counts will tend to have larger error.

5.4 Experiments

5.4.1 Experimental Results on the simulated Data

Table 5.1: ie vs. $\hat{ie}$ on the simulated data

model	0%	25%	50%	100%	200%
CCA	1.0000	0.9855	0.9461	0.8334	0.6384
	1.0000	0.9849	0.9440	0.8263	0.6268
	1.0000	0.9843	0.9414	0.8194	0.6114
weighted CCA	1.0000	0.9979	0.9931	0.9802	0.9523
	1.0000	0.9976	0.9917	0.9769	0.9469
	1.0000	0.9972	0.9903	0.9709	0.9393
LIN3	1.0000	0.9853	0.9457	0.8257	0.6063
	1.0000	0.9847	0.9441	0.8191	0.5920
	1.0000	0.9841	0.9410	0.8142	0.5751
Weighted LIN3	0.9998	0.9853	0.9449	0.8257	0.6051
	0.9998	0.9844	0.9424	0.8189	0.5889
	0.9998	0.9837	0.9395	0.8094	0.5733

Besides the real world dataset, we also generated simulated data on which to test our methods. The function of the simulated data is two fold: first, since we have already known every aspect of the dataset, the result for a specific model can be predicted theoretically, thus convince the others that the implementation detail is correct; second, with simulated data, we can test the intrinsic properties of models based from from different aspects. For every assumption, our simulated data have altogether 100 proteins, and each protein have 10 different peptide fragmentation sequences. the occurrence frequency of each single amino acid satisfy normal distribution with mean 4 and standard

Table 5.2: y vs. $\hat{y}$ on the simulated data

model	0%	25%	50%	100%	200%
CCA	0.9999	0.9977	0.9919	0.9669	0.8545
	0.9999	0.9972	0.9904	0.9590	0.8316
	0.9999	0.9965	0.9874	0.9471	0.8006
weighted CCA	0.9762	0.9738	0.9645	0.8135	0.3080
	0.9711	0.9703	0.9588	0.5050	0.2018
	0.9656	0.9649	0.9435	0.2240	0.0921
LIN3	1.0000	0.9980	0.9921	0.9684	0.8864
	1.0000	0.9975	0.9903	0.9627	0.8725
	1.0000	0.9968	0.9883	0.9519	0.8470
Weighted LIN3	1.0000	0.9978	0.9920	0.9668	0.8930
	1.0000	0.9975	0.9902	0.9619	0.8735
	1.0000	0.9968	0.9875	0.9552	0.8528

deviation 2. the input levels for different peptides are ranged from 20 to 250. every peptide fragmented from the same protein has the same input. Further more, to make the analysis more easy, we constraint every 10 proteins have the same input level. Thus, there are altogether 100 different input levels for all the peptides. To test the robustness of different models facing noise, we add Gaussian noise on the generated data. the noise has the mean 0 and the standard deviation can be adjusted, as we have shown in table 2 and table 3 in the following. Several noise levels ($d = 0\%, 25\%, 50\%, 100\%, 200\%$) were examined. The entire procedure is repeated $N = 100$ times. Each run yields a correlation coefficient for each predictor; the results are summarized by computing the median correlation coefficient for each predictor, the 75^{th} percentile, and the 25^{th} percentile over the 100 runs (Table 5.1 and Table 5.2). For this experiment, the CCA

model, the weighted CCA model, LIN3 and the weighted LIN3 were each tested to see the correlation coefficients on the simulated datasets. The corresponding experimental results are shown in Table 5.1 and Table 5.2). These tables show the correlation of observed and estimated values for ionization efficiency (ie v.s. $\hat{ie}$) and for spectral count (y v.s. $\hat{y}$). In general, we regard the latter correlation to be the more important one, since y is observed directly in an MS/MS experiment, whereas ie is not, but is computed from y using our estimates of the amounts of protein input.

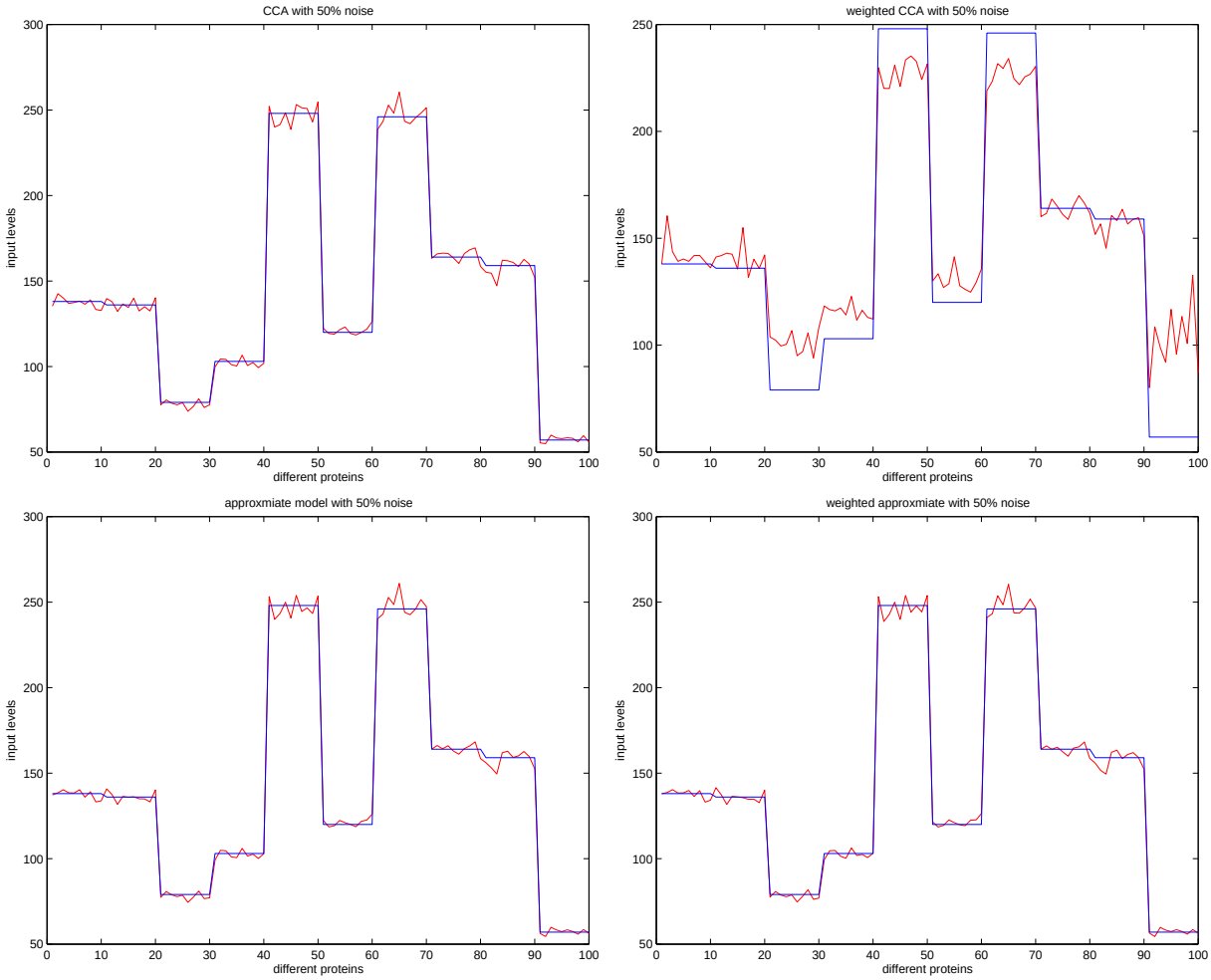

Figure 5.1: Illustration of the estimated input levels for different models with noise level 100%, they are CCA, weighted CCA, Approximate method, weighted approximated method respectively

From Tables 5.1 and 5.2, we can see that the correlation of y vs $\hat{y}$ is normally better than that of ie vs. $\hat{ie}$. Also, the results of LIN3 are very close to those of CCA methods, suggesting that it is a good approximation. More importantly, using y vs. $\hat{y}$, the LIN3 model performs much better than CCA, from which we can conclude that LIN3 model is more robust to noise than CCA. Similar results can also be seen in the real-world dataset, below. For both the CCA model and LIN3, the weighted strategy appears very effective, and performance can be improved greatly, which is evidence that the weighting strategy is robust for Gaussian noise.

Figure 5.4.1 shows the estimation of the input levels for different models, From which, we can see that all these four methods can effectively estimate the relative input levels of different peptides, but the weighted CCA model is more sensitive than the other three models. Also, the estimated values of the LIN3 model is close to those of the CCA model, which is also a good evidence that LIN3 is a good or even better approximation to the CCA model.

5.4.2 Experimental Results on Real-world Datasets

This section uses real-world data to experimentally evaluate the learning methods and models described above. The main evaluation strategy is ten-fold cross validation, with correlation coefficient used to measure the fit of a learned model to the testing portion of the data.

Study Design

We evaluated the learning methods on three data sets derived from tissue samples taken from Mouse. Similar in form to Table 2.1 in Chapter 2. For the learning methods developed in this chapter, each peptide must be represented as a vector, x. This section evaluates two ways of doing this, using vectors with 21 features and 421 features, respectively. The vectors with 21 features represent the amino-acid composition of a peptide.

Since there are twenty different amino acids, the vector has 20 features, $(x_1, ..., x_{20})$, where the value of feature x_i is the number of occurrences of a particular amino acid in the peptide. In addition, the vector has a 21^{st} feature, x_0, whose value is always 1, to represent a bias term, as is common in machine learning models [24]. The vectors with 421 features include the original 21 plus an additional 400 features representing the dimer composition of a peptide. A *dimer* is a sequence of two amino acids, and since there are 20 distinct amino acids, there are 400 distinct dimers.

We evaluated numerous combinations of feature vector, learning method and weighting scheme. Due to space limitations, we present only five of them here. In addition, because of the time required to execute CCA, we used it in only one combination: unweighted and with vectors having 21 features. We also evaluated four versions of the approximate method developed in Section 5.2. The first two versions are both unweighted and use vectors with 21 features and 421 features, respectively. We refer to these two versions as *LIN3-21* and *LIN3-421*. The other two versions are both weighted and use vectors with 21 features. The two weighting schemes used are $\mathbf{w}_i = ||Y_i||$ and $\mathbf{w}_i = 1/||Y_i||$, as described in Section 5.3. Finally, our methods are compared with the random models as proposed in Chapter 4.

Using ten-fold cross validation, we evaluated each of these five learning methods on each of the three Mouse data sets. Thus, each method was trained on nine tenths of the data (the training set), and the fitted model was then evaluated on the remaining one tenth of the data (the test set), and this was repeated in ten possible ways. Each training session produced an estimate, $\hat{\beta}$, of the parameter vector β, and an estimate, $\hat{in}$, of the input amount for each protein in the training set. Using $\hat{\beta}$, we estimated the ionization efficiency of each peptide in the entire data set, using the formula $\hat{ie} = x \bullet \hat{\beta}$, where x is the vector representation of the peptide. Applying univariate linear regression to Equation 5.5, we then estimated an input amount, $\hat{in}$, for each protein in the test set. We then estimated the spectral count of each peptide in the entire data set, using $\hat{y} = \hat{in} \cdot \hat{ie}$.

Finally, we compared the estimated and observed values of peptide spectral count (that is, $\hat{y}$ and y) by computing correlation coefficients.

The results are shown in Table 5.3. In this table, each column corresponds to a Mouse data set, and each row corresponds to a learning method. Each position in the table shows four numbers, stacked vertically. The top two numbers are the mean and standard deviation of the correlation coefficient of $\hat{y}$ and y on the training data. The bottom two numbers are the mean and standard deviation of the correlation coefficient on the test data. (Since at this stage, we are only interested in rough estimates of correlation coefficient, the ten estimates produced by ten-fold cross validation are enough.) In addition, by dividing $\hat{y}$ and y by $\hat{in}$, we get two different estimates of ionization efficiency, which we denote $\hat{ie}$ and ie, respectively. The correlation coefficient between these two estimates is what CCA tries to maximize. Thus, while the correlation coefficient of $\hat{y}$ and y measures the ability of the fitted model to predict experimental observations, the correlation coefficient of $\hat{ie}$ and ie provides the most direct measure of fit between the model and the data. The results are shown in Table 5.4, which has the same format as Table 5.3. Also, The performance of the these methods, along with the naive methods, is illustrated graphically in Figure 5.2, 5.3 which plot the mean value of Spearman rank correlation coefficients of 10-fold cross validation against the size of the feature vectors. Each point in the curve represents the mean Spearman correlation coefficient, while each curve in the figure represents a different method.

Results

The first point to notice is that of all the methods that use vectors with 21 features, CCA provides the best fit to the training data in Table 5.4. (only LIN3-421 produces a better fit, and only on the Kidney data, but it uses more features.) This is to be expected since CCA maximizes the correlation coefficient, which is what the table measures. On the other hand, LIN3-21 provides the best fit to the test data. The same is true in Table 5.3,

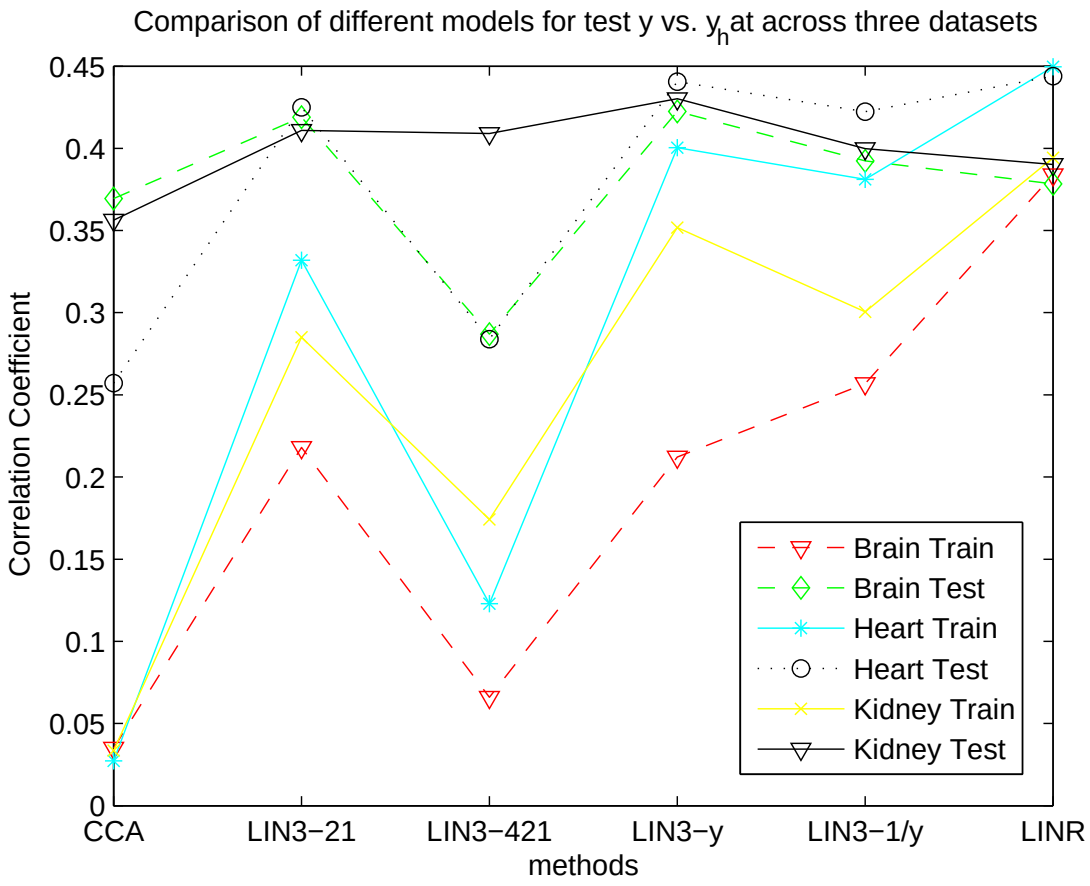

Figure 5.2: A comparison of the generalization performance of different models across the three datasets for y vs. $\hat{y}$

where LIN3-21 provides better test predictions of y, the peptide spectral count and the main biological observable. These results suggest that while our method may be an approximation of CCA, it may also be more appropriate for this problem, in terms of accuracy as well as speed.

The effect of the weighted methods is inconclusive. In Table 5.4, the unweighted LIN3-21 has consistently better performance on the test data than either of the two weighted schemes, but not dramatically better. In Table 5.3, the three methods perform comparably on the test data, though weights of $|1/y|$ seem to be marginally best, and weights of $|y|$ seems to be marginally worst, with LIN3-21 in between.

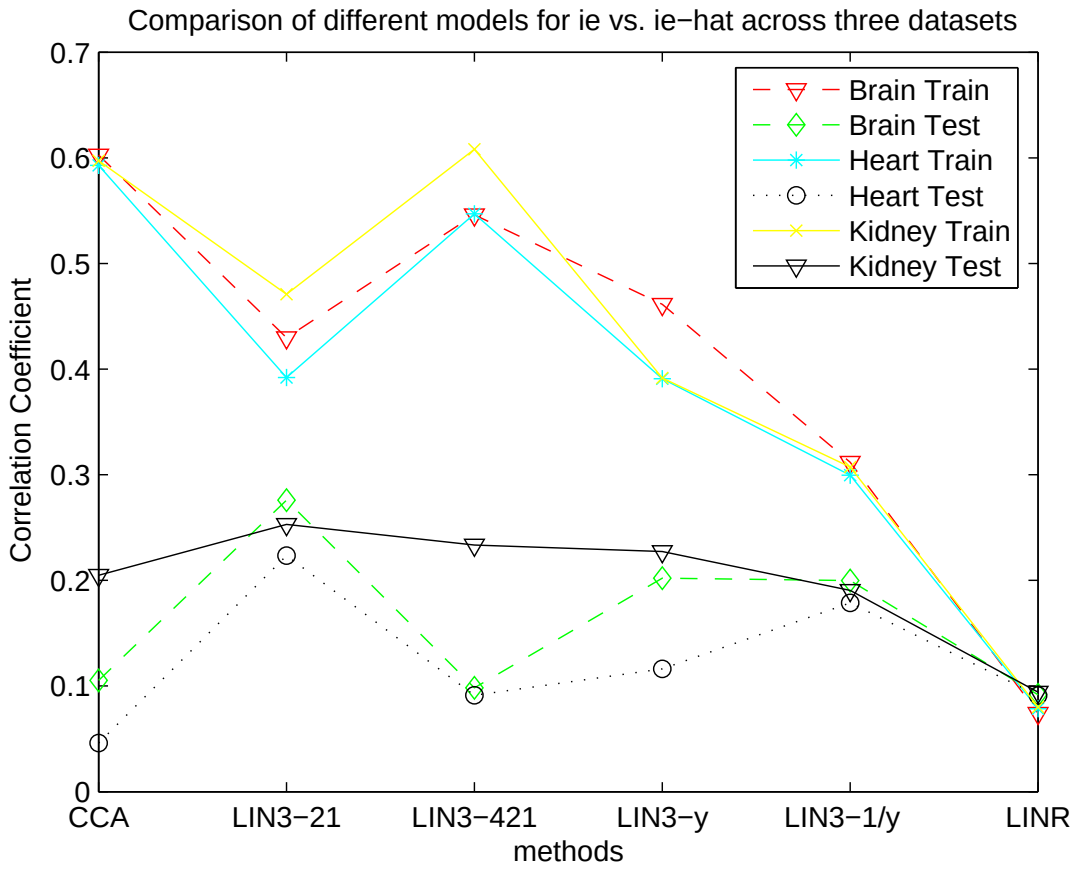

Figure 5.3: A comparison of the generalization performance of different models across the three datasets for ie vs. $\hat{ie}$

The effect of the larger feature vector is more conclusive. If we compare the LIN3-21 and LIN3-421 methods in Table 5.4, we can see that LIN3-421 shows evidence of overfitting, since the fit on the testing data is often much worse than on the training data. The 400 dimer values included among the 421 features thus appear to have little predictive value. Biologically, this a useful negative result.

Table 5.3 shows some apparently anomalous patterns. For instance, the fit on the testing data is often better than on the training data. Also, LIN3-21 has a better fit to the training data than LIN3-421, even though the features used in LIN3-21 are a subset of those used in LIN3-421. These patterns are probably a result of comparing $\hat{y}$ and y,

whereas the learning methods try to fit $\hat{ie}$ and ie. The same patterns are not present in Table 5.4, which compares $\hat{ie}$ and ie.

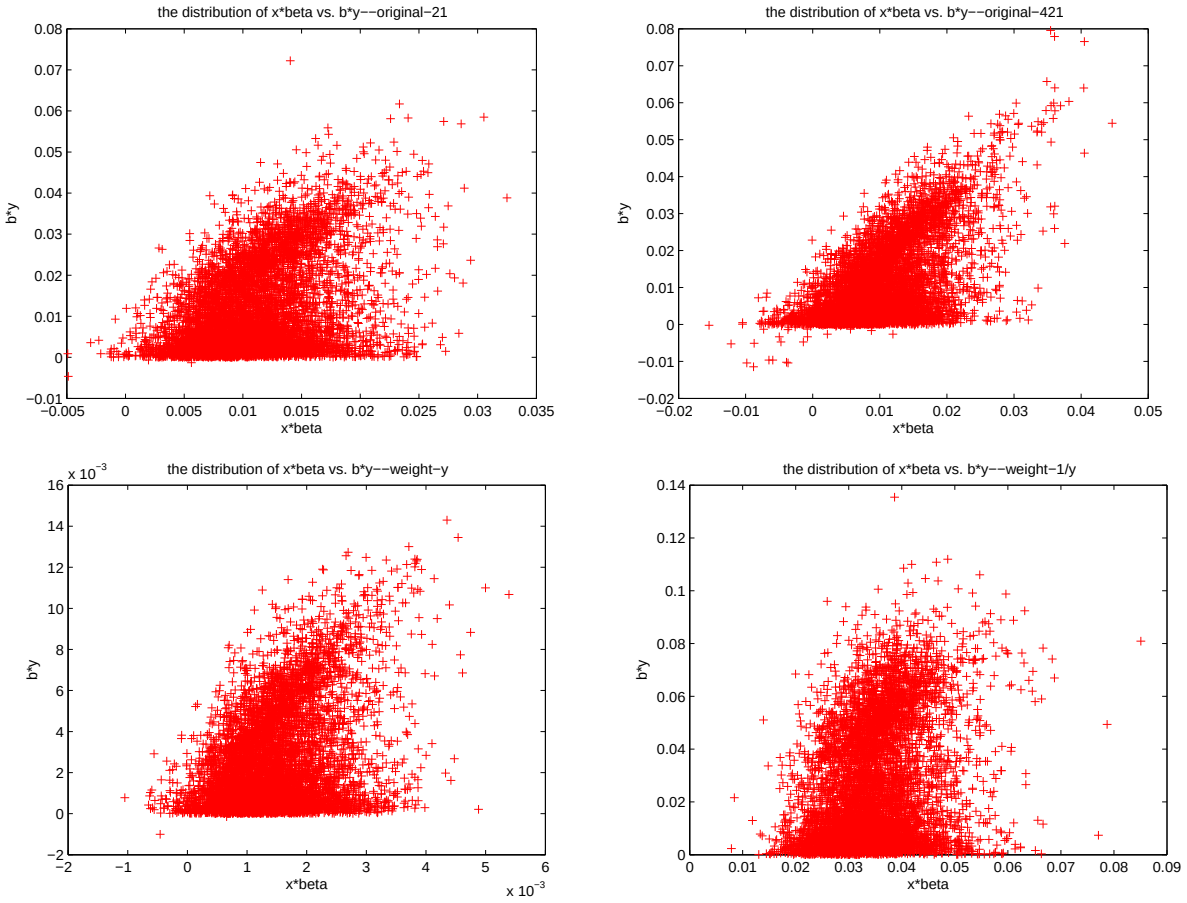

Figure 5.4: Plots of ie vs. $\hat{ie}$ as estimated by LIN3-21 (left) and LIN3-421 (right) on the Mouse Kidney data set.

In addition to the measurements presented in Tables 5.3 and 5.4, Figure 5.4 provides a visual representation of how well the estimated models fit the data. The figure shows two plots. In each plot, the horizontal axis is $\hat{ie}$, the vertical axis is ie, and each point represents a single peptide. The left plot was generated by the LIN3-21 method, and the right plot by the LIN3-421 method, both trained on the entire Kidney data set. The other methods generate similar plots. The first point to notice is that in both plots, the vast majority of values of $\hat{ie}$ and ie are positive, which is how things should be, since ionization

efficiency is inherently positive. The LIN3-421 method has more negative points than LIN3-21, but again, this could be a result of overfitting. The second point to notice is that each plot appears to consist of two components—a fairly linear diagonal component, and a less-linear horizontal blob. This suggests that there are two populations of peptides, those whose ionization efficiency is well-modeled by a linear function, and those whose ionization efficiency is much less predictable. This would explain why the correlation coefficients in Table 5.4 are low. It also suggests a natural topic for future research: characterizing those peptides that can be modeled linearly.

5.5 Conclusion of Canonical Correlation Analysis

In this chapter, CCA, LIN3 model as an approximate CCA and their application in mining peptide tandem MS/MS data were introduced. the main purpose of this work was to determine if the spectral counts of the MS/MS experiment could be predicated by some simple statistical models. Experiments on both simulated dataset and real-world datasets were conducted to evaluate the advantages and weakness of different models. Based on the experimental results, reasonable hypothesis were made which will be our future directions. Especially, even though CCA is too computational expensive to be practical, our proposed more efficient and robust approximation method LIN3 can be a useful data mining tool for a kind of interesting real-world problems, for which besides the corresponding coefficients for the predictors, there may also be different latent variables for responses. One of the limitations for LIN3 is that it's based on the specific structure of the matrix $\mathbf{Y}$ as introduced before. Even though for most spectrum based mining problem, this is a reasonable assumption, it makes LIN3 not as general purpose as CCA, for this, we are considering a technique by finding sub-clusters to capture the structure of the data. Thus reduce problem yields much smaller matrices of which the generalized eigenvalue decomposition and inverse can be computed more efficient for CCA.

Table 5.3: Correlation of y and $\hat{y}$ on real data

Method	Statistics	Brain Data	Heart Data	Kidney Data
CCA	Mean Train:	0.0350	0.0273	0.0335
	Std Train:	0.0292	0.0162	0.0413
	Mean Test:	0.3694	0.2575	0.3563
	Std Test:	0.1118	0.1683	0.0865
LIN3-21	Mean Train:	0.2180	0.3319	0.2850
	Std Train:	0.0356	0.0135	0.0583
	Mean Test:	0.4190	0.4250	0.4109
	Std Test:	0.1133	0.0906	0.0705
LIN3-421	Mean Train:	0.0660	0.1299	0.1742
	Std Train:	0.0529	0.1631	0.0745
	Mean Test:	0.2869	0.2839	0.4090
	Std Test:	0.1975	0.1098	0.0999
Weighted	Mean Train:	0.2121	0.4004	0.3518
LIN3-21	Std Train:	0.0678	0.0935	0.0363
$w = \lvert y \rvert$	Mean Test:	0.4225	0.4406	0.4302
	Std Test:	0.1004	0.0934	0.0951
Weighted	Mean Train:	0.2568	0.3811	0.3005
LIN3-21	Std Train:	0.0067	0.0168	0.0186
$w = \lvert 1/y \rvert$	Mean Test:	0.3924	0.4223	0.3999
	Std Test:	0.0924	0.0799	0.0586
LINR	Mean Train:	0.3839	0.4497	0.3943
	Std Train:	0.0062	0.0061	0.0059
	Mean Test:	0.3783	0.4440	0.3902
	Std Test:	0.0485	0.0479	0.0349

Table 5.4: Correlation of ie and $\hat{ie}$ on real data

Method	Statistics	Brain Data	Heart Data	Kidney Data
CCA	Mean Train:	0.6027	0.5929	0.5971
	Std Train:	0.0042	0.0054	0.0040
	Mean Test:	0.1054	0.0459	0.2049
	Std Test:	0.0763	0.0486	0.0846
LIN3-21	Mean Train:	0.4298	0.3921	0.4078
	Std Train:	0.0057	0.0081	0.0074
	Mean Test:	0.2759	0.2234	0.2530
	Std Test:	0.0432	0.0605	0.0485
LIN3-421	Mean Train:	0.5460	0.5469	0.6080
	Std Train:	0.1998	0.2616	0.0071
	Mean Test:	0.0982	0.0913	0.2335
	Std Test:	0.1125	0.1058	0.0424
Weighted	Mean Train:	0.4613	0.3909	0.3916
LIN3-21	Std Train:	0.0051	0.0104	0.0135
$w = \|y\|$	Mean Test:	0.2021	0.1163	0.2272
	Std Test:	0.0737	0.0788	0.0456
Weighted	Mean Train:	0.3118	0.2995	0.3072
LIN3-21	Std Train:	0.0056	0.0070	0.0093
$w = \|1/y\|$	Mean Test:	0.1999	0.1786	0.1905
	Std Test:	0.0349	0.0528	0.0376
LINR	Mean Train:	0.0746	0.0784	0.0802
	Std Train:	0.0144	0.0172	0.0127
	Mean Test:	0.0916	0.0911	0.0941
	Std Test:	0.0654	0.0259	0.0293

Chapter 6

Conclusions

This work uses data-mining techniques to take a first step towards developing a theory about modeling peptide tandem mass spectromety data. Specifically, to help to explain the difference between peptide spectral counts, we compare and evaluate different discriminant methods; To predict the peptide abundance and spectral count quantitatively, we develop three generative models of MS/MS data, and for each, we develop a family of methods for efficiently fitting the model to data. Because this is an initial study, the models were chosen for their simplicity and tractability, and the goal is to see how well (or poorly) they fit the data, and to quantify the error. Each model predicts the spectral count of a peptide based on two factors: its amino-acid sequence, and the abundance of the protein from which it was derived. The three models differ in their treatment of peptide ionization. However, they each provides an explanation for the linear relationship between protein abundance and spectral count. More importantly, we show how to use each model to estimate protein abundance from spectral count; Finally, we tried canonical correlation analysis model directly and showed its relationship with the LIN3 model we proposed before.

Appendix

Table 6.1: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Heart dataset with 22 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.1690	0.0244	0.1792	0.0507
LIN2	0.1854	0.0111	0.1599	0.0357
LIN3	0.4579	0.0055	0.4939	0.0524
INV1	0.3275	0.0062	0.3186	0.0429
INV2	0.5452	0.0048	0.5385	0.0472
INV3	0.4103	0.0058	0.5322	0.0508
EXP1	0.5566	0.0050	0.5495	0.0424
EXP2	0.5581	0.0050	0.5551	0.0449
LIND	0.4735	0.0061	0.4685	0.0599
INVD	0.5181	0.0061	0.5132	0.0536
EXPD	0.5178	0.0058	0.5137	0.0512
LINR	0.4486	0.0048	0.4436	0.0500
INVR	0.4553	0.0058	0.4523	0.0494
EXPR	0.5081	0.0050	0.5041	0.0479

Table 6.2: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Heart dataset with 22 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2624	0.0137	0.2648	0.0429
LIN2	0.2810	0.0174	0.2487	0.0501
LIN3	0.3682	0.0077	0.3180	0.0425
INV1	0.2566	0.0117	0.2579	0.0620
INV2	0.2350	0.0043	0.2310	0.0282
INV3	0.2625	0.0052	0.2216	0.0317
EXP1	0.3068	0.0050	0.2995	0.0325
EXP2	0.3076	0.0054	0.2949	0.0321
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0846	0.0119	0.0902	0.0593
INVR	0.0695	0.0104	0.0991	0.0274
EXPR	0.0548	0.0081	0.0500	0.0443

Table 6.3: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Heart dataset with 42 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.1509	0.0067	0.1699	0.0397
LIN2	0.1685	0.0078	0.1629	0.0553
LIN3	0.4622	0.0058	0.4926	0.0516
INV1	0.2580	0.0041	0.2525	0.0371
INV2	0.5438	0.0052	0.5328	0.0476
INV3	0.4096	0.0061	0.5274	0.0526
EXP1	0.5586	0.0051	0.5463	0.0425
EXP2	0.5364	0.0037	0.5266	0.0323
LIND	0.4735	0.0061	0.4685	0.0599
INVD	0.5181	0.0061	0.5132	0.0536
EXPD	0.5178	0.0058	0.5137	0.0512
LINR	0.4486	0.0048	0.4436	0.0500
INVR	0.4553	0.0058	0.4523	0.0494
EXPR	0.5081	0.0050	0.5041	0.0479

Table 6.4: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Heart dataset with 42 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2472	0.0093	0.2382	0.0449
LIN2	0.2349	0.0202	0.2168	0.0377
LIN3	0.3821	0.0069	0.3210	0.0419
INV1	0.2025	0.0054	0.2024	0.0297
INV2	0.2371	0.0059	0.2231	0.0266
INV3	0.2763	0.0057	0.2203	0.0330
EXP1	0.3141	0.0047	0.2921	0.0315
EXP2	0.3704	0.0067	0.3526	0.0386
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0846	0.0119	0.0902	0.0593
INVR	0.0695	0.0104	0.0991	0.0274
EXPR	0.0548	0.0081	0.0500	0.0443

Table 6.5: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Heart dataset with 62 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.1546	0.0146	0.1559	0.0423
LIN2	0.1800	0.0115	0.1681	0.0448
LIN3	0.3671	0.1686	0.4002	0.1583
INV1	0.2289	0.0084	0.2366	0.0501
INV2	0.3564	0.1660	0.3511	0.1684
INV3	0.4289	0.0060	0.5310	0.0489
EXP1	0.5687	0.0046	0.5531	0.0387
EXP2	0.1496	0.1183	0.1635	0.1083
LIND	0.4735	0.0061	0.4685	0.0599
INVD	0.5181	0.0061	0.5132	0.0536
EXPD	0.5178	0.0058	0.5137	0.0512
LINR	0.4486	0.0048	0.4436	0.0500
INVR	0.4553	0.0058	0.4523	0.0494
EXPR	0.5081	0.0050	0.5041	0.0479

Table 6.6: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Heart dataset with 62 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2726	0.0154	0.2807	0.0452
LIN2	0.2567	0.0365	0.2344	0.0567
LIN3	0.3832	0.0303	0.3249	0.0544
INV1	0.2433	0.0174	0.2490	0.0713
INV2	0.2256	0.0556	0.2218	0.0555
INV3	0.3021	0.0052	0.2452	0.0343
EXP1	0.3466	0.0055	0.3192	0.0325
EXP2	0.0874	0.0591	0.4418	0.0569
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0846	0.0119	0.0902	0.0593
INVR	0.0695	0.0104	0.0991	0.0274
EXPR	0.0548	0.0081	0.0500	0.0443

Table 6.7: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Heart dataset with 232 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.1253	0.0135	0.1290	0.0308
LIN2	0.1526	0.0112	0.1674	0.0658
LIN3	0.4423	0.0565	0.4303	0.0824
INV1	0.2703	0.0066	0.2634	0.0565
INV2	0.2159	0.0084	0.2092	0.0571
INV3	0.4419	0.0144	0.4963	0.0557
EXP1	0.6065	0.0042	0.5585	0.0427
EXP2	0.6169	0.0042	0.5684	0.0415
LIND	0.4735	0.0061	0.4685	0.0599
INVD	0.5181	0.0061	0.5132	0.0536
EXPD	0.5178	0.0058	0.5137	0.0512
LINR	0.4486	0.0048	0.4436	0.0500
INVR	0.4553	0.0058	0.4523	0.0494
EXPR	0.5081	0.0050	0.5041	0.0479

Table 6.8: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Heart dataset with 232 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2610	0.0132	0.2691	0.0410
LIN2	0.2701	0.0087	0.2935	0.0335
LIN3	0.4832	0.0331	0.3445	0.0733
INV1	0.2768	0.0101	0.2669	0.0551
INV2	0.2784	0.0173	0.3084	0.0429
INV3	0.4018	0.0068	0.3031	0.0441
EXP1	0.4612	0.0065	0.3951	0.0433
EXP2	0.4785	0.0058	0.4108	0.0403
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0846	0.0119	0.0902	0.0593
INVR	0.0695	0.0104	0.0991	0.0274
EXPR	0.0548	0.0081	0.0500	0.0443

Table 6.9: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Kidney dataset with 22 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.1515	0.0095	0.1540	0.0414
LIN2	0.2023	0.0160	0.2112	0.0666
LIN3	0.4188	0.0029	0.4678	0.0311
INV1	0.3189	0.0060	0.3191	0.0426
INV2	0.5181	0.0041	0.5130	0.0358
INV3	0.4041	0.0044	0.5117	0.0379
EXP1	0.5261	0.0037	0.5214	0.0303
EXP2	0.5309	0.0037	0.5254	0.0323
LIND	0.4222	0.0040	0.4215	0.0356
INVD	0.4721	0.0041	0.4695	0.0356
EXPD	0.4721	0.0075	0.2716	0.0357
LINR	0.3955	0.0054	0.3947	0.0342
INVR	0.4003	0.0047	0.3998	0.0404
EXPR	0.4610	0.0043	0.4579	0.0351

Table 6.10: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Kidney dataset with 22 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2479	0.0157	0.2449	0.0546
LIN2	0.2517	0.0191	0.2448	0.0631
LIN3	0.3657	0.0064	0.3488	0.0322
INV1	0.2855	0.0050	0.2874	0.0461
INV2	0.2769	0.0081	0.2660	0.0358
INV3	0.3199	0.0079	0.2813	0.0407
EXP1	0.3419	0.0066	0.3342	0.0258
EXP2	0.3576	0.0063	0.3489	0.0298
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0852	0.0161	0.0907	0.0435
INVR	0.0691	0.0144	0.0659	0.0314
EXPR	0.0524	0.0121	0.0737	0.0434

Table 6.11: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Kidney dataset with 42 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2160	0.0050	0.2186	0.0470
LIN2	0.1788	0.0169	0.1857	0.0805
LIN3	0.4231	0.0033	0.4685	0.0322
INV1	0.2803	0.0032	0.2784	0.0321
INV2	0.5171	0.0046	0.5049	0.0360
INV3	0.4115	0.0040	0.5098	0.0386
EXP1	0.5149	0.0038	0.5091	0.0330
EXP2	0.5364	0.0037	0.5266	0.0323
LIND	0.4222	0.0040	0.4215	0.0356
INVD	0.4721	0.0041	0.4695	0.0356
EXPD	0.4721	0.0075	0.2716	0.0357
LINR	0.3955	0.0054	0.3947	0.0342
INVR	0.4003	0.0047	0.3998	0.0404
EXPR	0.4610	0.0043	0.4579	0.0351

Table 6.12: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Kidney dataset with 42 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2810	0.0072	0.2717	0.0398
LIN2	0.2509	0.0253	0.2433	0.0713
LIN3	0.3763	0.0066	0.3490	0.0318
INV1	0.2274	0.0078	0.2309	0.0438
INV2	0.2784	0.0069	0.2548	0.0315
INV3	0.3344	0.0080	0.2812	0.0412
EXP1	0.2908	0.0050	0.2794	0.0327
EXP2	0.3704	0.0067	0.3526	0.0306
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0852	0.0161	0.0907	0.0435
INVR	0.0691	0.0144	0.0659	0.0314
EXPR	0.0524	0.0121	0.0737	0.0434

Table 6.13: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Kidney dataset with 62 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.1805	0.0176	0.1826	0.0433
LIN2	0.1652	0.0214	0.1685	0.0530
LIN3	0.4288	0.0032	0.4692	0.0313
INV1	0.2237	0.0127	0.2238	0.0400
INV2	0.3119	0.1464	0.3198	0.1183
INV3	0.4177	0.0038	0.5101	0.0354
EXP1	0.5368	0.0035	0.5240	0.0300
EXP2	0.1597	0.1055	0.1855	0.1176
LIND	0.4222	0.0040	0.4215	0.0356
INVD	0.4721	0.0041	0.4695	0.0356
EXPD	0.4721	0.0075	0.2716	0.0357
LINR	0.3955	0.0054	0.3947	0.0342
INVR	0.4003	0.0047	0.3998	0.0404
EXPR	0.4610	0.0043	0.4579	0.0351

Table 6.14: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Kidney dataset with 62 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2927	0.0195	0.2742	0.0425
LIN2	0.2463	0.0205	0.2402	0.0514
LIN3	0.3868	0.0067	0.3544	0.0323
INV1	0.2452	0.0276	0.2717	0.0443
INV2	0.2538	0.0338	0.2567	0.0452
INV3	0.3569	0.0052	0.2933	0.0334
EXP1	0.3694	0.0052	0.3465	0.0309
EXP2	0.0805	0.0408	0.3532	0.0518
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0852	0.0161	0.0907	0.0435
INVR	0.0691	0.0144	0.0659	0.0314
EXPR	0.0524	0.0121	0.0737	0.0434

Table 6.15: Spearman rank correlation coefficients about y vs. $\hat{y}$ on the Mouse Kidney dataset with 232 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2442	0.0443	0.2299	0.0672
LIN2	0.1851	0.0142	0.1884	0.0501
LIN3	0.2181	0.0829	0.2027	0.1127
INV1	0.1979	0.0175	0.1826	0.0501
INV2	0.2189	0.0233	0.2299	0.0435
INV3	0.1652	0.1047	0.2314	0.1665
EXP1	0.4128	0.1700	0.3894	0.1602
EXP2	0.5872	0.0034	0.5338	0.0289
LIND	0.4222	0.0040	0.4215	0.0356
INVD	0.4721	0.0041	0.4695	0.0356
EXPD	0.4721	0.0075	0.2716	0.0357
LINR	0.3955	0.0054	0.3947	0.0342
INVR	0.4003	0.0047	0.3998	0.0404
EXPR	0.4610	0.0043	0.4579	0.0351

Table 6.16: Spearman rank correlation coefficients about ie vs. $\hat{ie}$ on the Mouse Kidney dataset with 232 features (10-cross validation settings)

Models	mean train	std dev-train	mean test	std dev-test
LIN1	0.2684	0.0434	0.2591	0.0623
LIN2	0.3001	0.0161	0.2823	0.0431
LIN3	0.4439	0.0091	0.3015	0.0477
INV1	0.1929	0.0732	0.1847	0.1153
INV2	0.2586	0.0610	0.2589	0.0804
INV3	0.4322	0.0057	0.2752	0.0519
EXP1	0.4250	0.0401	0.4062	0.0559
EXP2	0.4827	0.0060	0.4072	0.0387
LIND	NA	NA	NA	NA
INVD	NA	NA	NA	NA
EXPD	NA	NA	NA	NA
LINR	0.0852	0.0161	0.0907	0.0435
INVR	0.0691	0.0144	0.0659	0.0314
EXPR	0.0524	0.0121	0.0737	0.0434

Bibliography

[1] Aebersold, R., Mann, M. *Mass spectrometry-based proteomics*, Nature 422, pp 198-207. 2003

[2] Becker, S. *Mutual information maximization: models of cortical self-organization.* Network: Computation in Neural Systems, 7, pp 7-31. 1996

[3] Bickel, P., Doksum, K. *Mathematical Statistics: Basic Ideas and Selected Topics*, Holden-Day INC. 1977

[4] Bonner, A., Liu, H. *Comparison of Discrimination Methods for Peptide Classification in Tandem Mass Spectrometry.* 2004 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (IEEE CIBCB'04). 2004

[5] Bonner, A., Liu, H. *Predicting Protein Levels from Tandem Mass Spectrometry Data.* NIPS-04 Workshop on New Problems and Methods in Computational Biology. Neural Information, Neural Information Processing Systems 17 (NIPS'04). 2004

[6] Bonner, A., Liu, H. *Development and Evaluation of Methods for Predicting Protein Levels and Peak Intensities from Tandem Mass Spectrometry Data* . Technical Report, Department of Computer Science, University of Toronto. 2004

[7] Bonner, A., Liu, H. *Canonical Correlation, an Approximation, and its Application to the Mining of Peptide Tandem Mass Spectrometry Data.* Submitted to SDM workshop on mining Scientific Database. 2005

[8] Bornsen, K.O., Gass, M.A., Bruin, G.J., Adrichem, J.H.et al. *Influence of solvents and detergents on matrix-assisted laser desorption/ionization mass spectrometry measurements of proteins and oligonucleotides*Rapid Commun Mass Spectrom 11, pp 603-609. 1997

[9] Breiman, L., Friedman, J., Olshen, R., and Stone, C. J., *Classification and Regression Trees*,, Chapman and Hall. 1984

[10] Corthals, G.L., Wasinger, V.C., Hochstrasser, D.F.and Sanchez, J.C. *The dynamic range of protein expression: a challenge for proteomic research*,Electrophoresis 21, pp 1104-1115. 2000

[11] David, S.W., *Fundamentals of Matrix Computation*, Wiley-Interscience. 2002

[12] Dogruel, D., Nelson, R.W., Williams, P., *Peptide characterization using bioreactive mass spectrometer probe tips*, Rapid Commun Mass Spectrom 2, pp 695-700. 1998

[13] Duda, R., Hart, P., Stork, D. *Pattern Classification*, second edition. Wiley-Interscience, 2001.

[14] Dudoit, S., Fridlyand,J. and Speed,T.P., *Comparison of discrimination methods for the classification of tumors using gene expression data.* Journal of the American Statistical Association 97(457), pp 77-87. 2002

[15] Elias, J.E., Gibbons,F.D. King,O.D. Roth,F. Gygi,S.P. *Intensity-based protein identification by machine learning from a library of tandem mass spectra*,Nature, biotechnology. Volume 22 Number 2, pp14-219. 2004

[16] Eng, J., McCormack, A., Yates, J.R. *An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database*,j.Am.Soc.Mass Spectrom.5,, pp 976-989, 1994

[17] Fieguth, P.W., Irving, W. W. and Willsky, A. S. *Multiresolution model development for overlapping trees via canonical correlation analysis.* In International Conference on Image Processing, pp 45-48. 1995.

[18] Figueroa, I.D., Torres, O., Russell, D.H., *Effects of the Water Content in the Sample Preparation for MALDI on the Mass Spectra*, Annal Chem. 70, pp 4527-4533. 1998

[19] Fisher, R.A. *The used of multiple measurements in taxonomic problems*, Annals of Eugenics, 7, pp 179-188. 1936

[20] Gaskell, S. *Electrospray: Principles and practices.* Journal of Mass Spectrometry, 32, pp 677-688. 1997

[21] Gay, S., Binz, P.A., Hochstrasser, D.F., Appel, R.D. *Peptide mass fingerprinting peak intensity prediction: extracting knowledge from spectra*, Proteomics 2, pp 1374-1391. 2002

[22] Gygi,S.P., Corthals, G.L., Zhang, Y., Rochon, Y., and Aebersold, R. *Evaluation of two-dimensional gel electrophoresis-based proteome analysis technology*, Proc.Natl.Acad.Sci.pp 9930-9395. 2000

[23] Gygi, S.P., Rist, B., Gerber, S.A.,Turecek, F., Gelb, M.H. and Aebersold, R. *Quantitative analysis of complex protein mixtures using isotope-coded affinity tags*, Nature Biotechnology 17(10), pp 994-999. 1999

[24] Hastie, H., Tibshirani, R., Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer Series in Statistics, 2001.

[25] Hotelling, H. *Relations between two sets of variates.* Biometrika, 28: pp 321-377. 1936.

[26] Jordan, M. *Why the Logistic Function? A tutorial discussion on probabilities and neural networks.* Computational Cognitive Science Technical Report 9503. Massachusetts Institute of Technology. 1995.

[27] Kay, J. *Feature discovery under contextual supervision using mutual information.* In International Joint Conference on Neural Networks, volume 4, pp 79-84. 1992.

[28] Knochenmuss, R., Karbach, V.,Wiesli, U., Breuker, K.,Zenobi, R., *The matrix suppression effect in matrix-assisted laser desorption/ionization: Application to negative ions and further characteristics*, Rapic Commun.Mass Spectrom.12, pp 529-534, 1998

[29] Kocher, F., Favre, A, Gonnet, F., Tabet, J.C.et al., *Proton affinities of the commonly occurring L-amino acids by using electrospray ionization-ion trap mass spectrometry.* Jouranl of Mass Spectrom.33, pp 921-935. 1998

[30] Kussmann, M., Nordhoff, E., Rahbek-Nielsen, H., Haebel, S.et al., *Matrix-assisted laser desorption/ionization mass spectrometry sample preparation techniques designed for various peptide and protein analytes,* Journal of Mass Spectrom. 32 pp 593-601. 1997

[31] Legendre, P., Lapointe, F.J., Casgrain, P. *A classification of pure malt Scotch whiskies.* Applied Statistics 43: pp 237-257. 1984

[32] Liebler, D.C. *Introduction to Proteomics, tools for the new biology*, Humana Press, NJ. 2002.

[33] Liu, H., Sadygov, R.G. and Yates, J.R. *A Model for Random Sampling and Estimation of Relative Protein Abundance in Shotgun Proteomics*, Anal. Chem. 76, pp 4193–4201. 2004.

[34] McLachlan, G.J. *Discriminant analysis and Statistical pattern recognition.* Wiley,New York. 1992.

[35] Mann, M., Hendrickson, R.C., Pandey, A. *Analysis of proteomes by mass spectrometry*, Annu.Rew.Biochem.70, pp 437-473. 2001

[36] Mardia, K.V., Kent, J.T. and Bibby, J.M. *Multivariate Analysis*. Academic Press, Inc. 1979

[37] Martin, K., Spickermann, J., Rader, H.J., Mullen, K.,*MALDI-TOF mass spectrometry in polymer analytics*, Rapid Commun Mass Spectrom.10, pp1471-1474. 1996

[38] Mathurin, J.C., Gregoire, S.,Brunot, A. Tabet, J.C.et al. *An investigation of ion/ion interactions which arise during the simultaneous confinement of positive ions and negative ions in an ion trap mass spectrometer*, Journal of Mass Spectrom.32, pp 829-837, 1997

[39] Ong,S., Blagoev,B., Kratchmarovat, I., Kristensen, D.B., Steen, H., Pandey, A. and Mann, M. *Stable Isotope Labelling by Amino Acid in Cell Culture, SILAC, as a Simple and Accurate Approach to Expression Proteomics*, Molecular & Cellular Proteomics, pp 376-386. 2002.

[40] Pachter, L., Alexandersson, M., Cawley, S. *Applications of Hidden Markov Models to Alignment and Gene Finding Problems*, Proceedings of the Fifth Annual Conference on Computational Biology, pp 241–248. 2001

[41] Pandey, A., Mann,M. *Protemics to study genes and genomes*, Nature 405, pp 837-846, 2000

[42] Poritz, A.B. *Linear Predictive Hidden Markov Models and the Speech Signal*, Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, pp 1291–1294. 1982.

[43] Purves, R. W., Gabryelski, W., Li, L., *Investigation of the Quantitative Capabilities of an Electrospray Ionization Ion-Trap/Linear Time-of-Flight Mass Spectrometer*, Rapic Commun.Mass Spectrom.2, pp 695-700. 1998

[44] Rice, J. *Mathematical Statistics and Data Analysis*, second edition. Duxbury Press. 1995.

[45] Ripley, B.D. *Pattern recognition and neural networks* Cambridge University Press, Cambridge, New York. 1996

[46] Siuzdak, G. *The Expanding Role of Mass Spectrometry in Biotechnology*, Mcc Press. 2003.

[47] Thosma, K., Khaled, R.,Dragan, R., *PRISM, a Generic Large Scale Proteomics investigation Strategy for Mammals*, Molecular& Cellular Proteomics, pp 96-106. 2003

[48] Washburn, M.P., Wolters, D., and Yates, J.R., III *Large-scale analysis of the yeast proteome by multidimensional protein identification technology*, Nat. Biotechnol. 19, pp 242-247. 2002