

2009

## A Bayesian network approach to feature selection in mass spectrometry data

Karl W. Kushner  
*College of William & Mary - Arts & Sciences*

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Bioinformatics Commons](#), and the [Mathematics Commons](#)

---

### Recommended Citation

Kuschner, Karl W., "A Bayesian network approach to feature selection in mass spectrometry data" (2009). *Dissertations, Theses, and Masters Projects*. Paper 1539623543.  
<https://dx.doi.org/doi:10.21220/s2-fjc0-7z38>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact [scholarworks@wm.edu](mailto:scholarworks@wm.edu).

A Bayesian Network Approach to Feature Selection  
in Mass Spectrometry Data

Karl Wayne Kuschner

Williamsburg, Virginia

MS, National Security Studies, Naval War College, 1996  
MA, Physics, University of Texas at Austin, 1993  
BS, Mathematics and Physics, U.S. Air Force Academy, 1983

A Dissertation presented to the Graduate Faculty  
of the College of William and Mary in Candidacy for the Degree of  
Doctor of Philosophy

Department of Physics

The College of William and Mary  
May 2009

## APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of  
the requirements for the degree of

Doctor of Philosophy



---

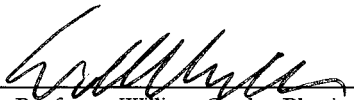
Karl Wayne Kuschner

Approved by the Committee, April, 2009



---

Committee Chair  
Chancellor Professor Eugene Tracy, Physics  
The College of William and Mary

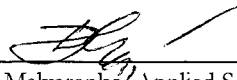
---

Professor William Cooke, Physics  
The College of William and Mary



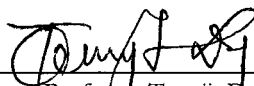
---

Professor John Delos, Physics  
The College of William and Mary



---

Dr. Dariya Malyarenko, Applied Science  
The College of William and Mary



---

Assistant Professor Tanujit Dey, Mathematics  
The College of William and Mary

## ABSTRACT PAGE

One of the key goals of current cancer research is the identification of biologic molecules that allow non-invasive detection of existing cancers or cancer precursors. One way to begin this process of *biomarker discovery* is by using time-of-flight mass spectroscopy to identify proteins or other molecules in tissue or serum that correlate to certain cancers. However, there are many difficulties associated with the output of such experiments. The distribution of protein abundances in a population is unknown, the mass spectroscopy measurements have high variability, and high correlations between variables cause problems with popular methods of data mining. To mitigate these issues, Bayesian inductive methods, combined with non-model dependant information theory scoring, are used to find feature sets and build classifiers for mass spectroscopy data from blood serum. Such methods show improvement over existing measures, and naturally incorporate measurement uncertainties. Resulting Bayesian network models are applied to three blood serum data sets: one artificially generated, one from a 2004 leukemia study, and another from a 2007 prostate cancer study. Feature sets obtained appear to show sufficient stability under cross-validation to provide not only biomarker candidates but also families of features for further biochemical analysis.

## TABLE OF CONTENTS

List of Figures .....	iv
Acknowledgments.....	vi
Chapter 1: Introduction .....	1
Background .....	1
Mass Spectrometry.....	1
Goal .....	3
Chapter 2: Mathematical Tools.....	7
Notation .....	7
Product and Sum Rules of Probability.....	8
Probability Distributions.....	8
Conditional Probabilities .....	8
Information Entropy.....	9
Mutual Information and Conditional Mutual Information.....	11
Bayes' Theorem .....	13
Chapter 3: Data Sets and Signal Processing.....	17
Sample Collection and Bias Avoidance.....	17
Leukemia Data.....	19
Generated Data .....	21
Prostate Cancer Data.....	26
Signal Processing .....	28
Data Analysis .....	28
Normalization.....	31
Chapter 4: Classification and Feature Selection.....	34
Feature Set Selection.....	36
Filter and Wrapper techniques .....	36
Wrapper methods .....	37
Naïve Bayesian Classifiers.....	38
Bayesian Networks.....	42
Bayesian Networks and Causality.....	47
Bayesian Classifier Construction .....	48
Structure Learning.....	49
Parameter Learning.....	51
Mutual Information with Class.....	54
Discretization .....	55
Cross-Validation .....	56
Chapter 5: Application of the Naïve Bayesian Classifier .....	60
Classification performance .....	61

Feature Selection .....	64
Error Rates .....	65
Feature Set Stability.....	67
Effect of Correlations.....	69
Correlation Effects.....	71
Chapter 6: Bayesian Network Algorithm.....	75
Goal .....	75
Algorithm.....	76
Initial data processing.....	77
Structure Learning.....	77
Mutual Information .....	79
Adjacency Matrix.....	82
First Level Connections.....	87
Second Level Connections.....	87
Parent-Child Identification.....	88
Metavariables .....	91
Parameter Learning.....	93
Classification and Error Rates .....	95
Chapter 7: Results and Analysis.....	96
Generated Data .....	97
Naïve Bayesian Classifier.....	97
Bayesian Network .....	101
Analysis .....	106
Leukemia Data.....	108
Naïve Bayesian Classifier.....	108
Bayesian Network .....	112
Analysis .....	117
Prostate Cancer Data.....	119
Naïve Bayesian Classifier.....	119
Bayesian Network .....	122
Analysis .....	127
Chapter 8: Conclusion.....	128
Appendix A: Mathematics .....	130
Maximum Entropy.....	130
Maximum Mutual Information.....	131
Naïve Bayesian Classifier Instability .....	132
Appendix B: MATLAB Code.....	135
Naïve Bayesian Classifier Code .....	135
Bayesian Network Algorithm.....	146
Code for Creating Generated Data.....	171

Appendix C: Results .....	175
Index .....	179
Glossary .....	180
Works Cited.....	182
Vita .....	185

## LIST OF FIGURES

<i>Number</i>	<i>Page</i>
Figure 1: A portion of a typical mass spectrum.....	2
Figure 2: Leukemia data .....	20
Figure 3: Mislabeled replicate spectra .....	21
Figure 4: Generated data distribution for highly diagnostic peak .....	23
Figure 5: Distribution for highly diagnostic peak after de-normalization .....	25
Figure 6: Generated data.....	26
Figure 7: PCA data .....	27
Figure 8: The need for signal processing .....	30
Figure 9: Histogram of normalization factors.....	33
Figure 10: Bayesian network .....	42
Figure 11: Causal chain for disease .....	47
Figure 12: Sample population and instrument function .....	53
Figure 13: Mutual information threshold .....	55
Figure 14: Naïve Bayesian classifier .....	60
Figure 15: Nominal classification error rates of individual variables.....	63
Figure 16: Cross-validated error rate, forward selection, PCA data .....	65
Figure 17: Error rate during backward elimination, Leukemia data .....	66
Figure 18: Variable selection frequency, 5 forward selection trials .....	67
Figure 19: Correlation between two diagnostic peaks.....	70
Figure 20: Generated data set with perfectly correlated features .....	71
Figure 21: Generated data set with correlated features with noise .....	72
Figure 22: Bayesian network for MS data.....	75
Figure 23: Histogram of MI between features and the class.....	80
Figure 24: Mutual information between variables, QC data .....	81
Figure 25: MI threshold effects under 10-fold cross-validation .....	82
Figure 26: Adjacency matrix representation.....	83
Figure 27: Results of optimizing discretization boundaries.....	84
Figure 28: Center bin isolates uncertainty.....	85
Figure 29: Search for optimal boundaries for three bin discretization.....	86
Figure 30: Removal of false connection to class .....	88
Figure 31: Effect of increasing drop threshold.....	90
Figure 32: Abundance probability differences by class .....	94
Figure 33: Error rate during forward selection, generated data.....	98
Figure 34: Error rate during feature selection, generated data.....	100
Figure 35: Distribution of features 3 and 4, generated data .....	101
Figure 36: Effect of MI Threshold .....	102
Figure 37: Frequency of class-variable connections, generated data .....	104
Figure 38: Error rate distribution, generated data .....	106
Figure 39: Resulting Bayesian network, generated data .....	107
Figure 40: Error rate during forward selection, Leukemia data.....	108



Figure 41: Error rate during feature selection, Leukemia data .....	110
Figure 42: Frequency of class-variable connections, Leukemia data .....	113
Figure 43: Histogram of CV error rates, Leukemia data.....	116
Figure 44: Resulting Bayesian network, Leukemia data .....	117
Figure 45: Leukemia (red) and normal spectra, vicinity feature 198.....	118
Figure 46: Error rate during forward selection, PCA data .....	119
Figure 47: Error rate during repeated forward selection, PCA data.....	120
Figure 48: Error rate during backward elimination, PCA data .....	121
Figure 49: Error rate during feature selection, PCA data .....	122
Figure 50: MI threshold effects under 10-fold cross-validation .....	123
Figure 51: Effect of increasing drop threshold, PCA data .....	124
Figure 52: Error rates from the BN algorithm, PCA data.....	127
Figure 53: Distribution of 11.7 kDa peak by class, Leukemia data.....	128

## ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Dr. Gene Tracy, Dr. Dasha Malyarenko, and Dr. Bill Cooke for their guidance and support during the development of this project. Funding for this project was provided in part by the National Cancer Institute under grant R01-CA126118.

## CHAPTER 1: INTRODUCTION

### Background

In early 2001, Incogen, a bioinformatics company, received funding from the Commonwealth of Virginia to move to Williamsburg and lead a bioinformatics consortium that included the College of William and Mary, among others. The next year, Incogen began a Small Business Innovation Research project, funded by the National Institute of Health. This project added Eastern Virginia Medical School in Norfolk, Virginia as a collaborator, and expanded the consortium's previous work to develop a set of computational tools for classifying biologic data, including data derived from mass spectrometry (MS).

That work led to an ongoing project whose goal is to create tools for “computationally improved signal processing for mass spectrometry data.” One of the steps in that project, and the focus of this work, is the development of methods to exploit the improved MS data to find biologically relevant information.

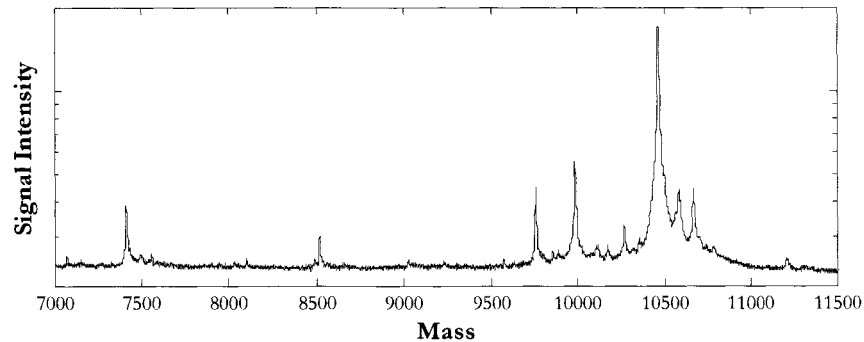
### Mass Spectrometry

A mass spectrometer is an instrument that takes some sample of material, biologic or otherwise, and measures the relative amounts of constituent materials—ordered by molecular mass<sup>1</sup>—in the sample. The output, shown in Figure 1, is called a *mass*

---

<sup>1</sup> Actually, mass divided by net charge of the molecule; see next section.

*spectrum* and is initially continuous<sup>2</sup> in nature, with very low signals representing mass regions where nothing was found, and spike-shaped structures (called “peaks”) representing a relatively large amount of material at a particular mass. The signal intensity is shown here on a logarithmic scale, but in arbitrary units. The horizontal axis values are the atomic weight being detected.



**Figure 1: A portion of a typical mass spectrum**

One type of mass spectrometry instrument works by ionizing the molecules in a sample, typically by an intense laser pulse or ion collision, then accelerating the resulting ions through an electric potential of a few kV. After the molecules have been accelerated to some terminal velocity  $v$ , which depends on their mass  $m$  and electric charge  $z$  as well as the electric potential  $V$ , they float down a field-free time of flight (TOF) tube and strike a detector. The energy  $E$  gained relates the electric potential and velocity by  $E = zV = \frac{1}{2}mv^2$ . Low mass ions reach a higher velocity and hence strike the detector first; heavy ions are detected last. By measuring the number of detections along a time scale, then converting the time axis into mass per unit charge ( $m/z$ ), a spectrum of signal intensity vs.  $m/z$  is created. While this

---

<sup>2</sup> Insomuch as each time point has a corresponding integer number of detections, the spectrum is actually discrete; on the scale of the entire spectrum, it is, for all intents, continuous.

is not the only method of mass spectrometry, it is a common one used in the field of proteomics. Its ability to survey a wide range of mass values aids the search for important proteins, as opposed to other methods, which might search for the abundance of a material at a specific  $m/z$  value.

Our group has data available from two types of TOF-MS instruments: matrix-assisted laser desorption/ionization (MALDI), and surface-enhanced laser desorption/ionization (SELDI), which is a special type of MALDI.

There are several errors associated with this type of instrument. Although we would like the peaks to be infinitely narrow “spikes,” they in fact have finite width due to the method of ionization and detection. In addition, the time that a specific molecule arrives differs slightly from trial to trial, and the intensity measured can vary for reasons other than true abundance variations in the sample. Another important error arises because of the violence of the initial ionization and the several ways a single molecule can show up—with charge  $z > 1$  (called multiply-charged states), in fragments, or with small common molecules such as the chemical matrix attached (adducts) or detached (neutral loss). These processes result in peaks at different  $m/z$  values that actually represent a single underlying molecule.

## Goal

Early detection of cancer dramatically increases the long-term survival rate of those afflicted [1]. However, most cancers are difficult to detect early, and accurate

analysis often requires surgery or biopsy followed by forensic pathology of the tissue.

The use of mass spectroscopy to search for biological markers, or biomarkers, in easily obtained biologic samples would enable higher throughput and less invasive testing. Since cancers typically cause variations in the gene expression—and hence protein abundance—of affected cells, researchers hope to find traces of these over or under expressed (or mutated) proteins that would differentiate samples from those with, and those without, the disease. Blood serum (blood with cells and platelets removed) is one of the easiest samples to obtain, and if the protein markers can be found to be transported in the blood, a test for early detection could be designed.

The difficult part of this task is the detection and identification of the biomarker. There are some 30,000 or more genes in the human genome, which express at least 100,000 different proteins varying across twelve orders of magnitude in abundance – far beyond the resolution of current instruments. In addition, the natural variation of protein abundance across a population is often wide, and can mask any variation between sub-groups, such as those with or without a disease. Even a single individual has a dynamic proteome; the blood serum changes throughout the day as food is digested and proteins are absorbed in the body—one of our colleagues at EVMS is able to determine whether a patient has eaten recently simply by the opacity of a vial of blood.

Our group has found that one of the largest errors in the process stems from the chemical and physical preparation of the samples prior to the MS measurements [2]. We have noted, for example, coefficients of variation (CV) of 5% from a single robotically prepared sample that is measured multiple times, and a CV of 30-40% from a single serum sample that is robotically prepared into multiple instrument samples prior to measurement. This is not a condemnation of the robotic process over manual preparation; in fact, the opposite is true—manual chemical preparation will introduce even more variation. The biochemical preparation steps, such as the amount of materials mixed, introduce this variability. To date, the problem of correcting errors associated with the biochemistry of sample preparation have been somewhat intractable, but our group continues to innovate in this area.

Other than the preparation protocols, there are three main areas where we seek improvement in the current technology—more accurate and precise measurement of the samples by improvements in the MS instrumentation, better analysis of the spectra produced (via noise reduction and other signal processing), and finally, better methods of mining the data for the biomarkers.

Current preparation and instrumental errors, such as those described above, make biomarker identification part art and part science. No single, well-accepted, and successful methodology for data analysis and biomarker discovery yet exists. In fact,

Proteomics methods based on mass spectrometry hold special promise for the discovery of novel biomarkers that might form the foundation for new clinical blood tests, but to date their contribution to the diagnostic armamentarium has been disappointing [3].

The MS group at William and Mary is pursuing improvements in all three areas, but it is the last goal—improved data mining—that is the focus of this work. Specifically, the research described herein seeks to find molecules that are diagnostic of the disease state, discard those that are not, and determine the data-derived relationships among these candidates. In addition, we want to arrange the selected variables into a stable classifier that is predictive when new data is introduced.



## CHAPTER 2: MATHEMATICAL TOOLS

### Notation

Much of the mathematics used in this work is from the field of probability theory.

As is common, capital letters represent statements, such as  $A =$  “the patient has the disease” or  $X =$  “the signal intensity is between 100 and 110.”

The function  $P(A)$  represents the probability that  $A$  is true, or, more informally, can take on one of the values  $A=a$ . A vertical bar after a capital letter, followed by one or more capital letters, represents conditions that are assumed to be true prior to the evaluation of the unknown, hence  $P(A|B)$  is read “the probability of  $A$ , given that  $B$  is true.” This is known as a conditional probability.

As alluded to previously, small letters denote the values of a variable represented by its capital letter, so that one would write “the probability that  $X=x$  is true, given that  $Y$  has the value  $y$ ” as  $P(X=x|Y=y)$ , or often  $P(x|y)$ . These values typically represent measurements, and a set of such values  $\{x_1, x_2, x_3, \dots, x_n\}$  is written as the bold  $\mathbf{x}$  and called a case.

A conjunction, or logical “and,” is denoted by a comma, or if the meaning is clear, two statements joined, e.g.  $(A \text{ AND } C) = (A,C) = (AC)$ . A logical “or” is always represented by a plus sign between statements, as in  $(A \text{ OR } B) = (A+B)$ . A tilde appearing before a capital letter represents negation, so that  $\text{NOT } A = \sim A$ .

## Product and Sum Rules of Probability

Two rules of the algebra of probability theory that we will use most often are termed the product rule and sum rule. More information (and a proof) can be found in Jaynes, 2003 [4]. They are

$$\text{(Product Rule)} \quad P(A, B) = P(A|B)P(B) \quad (1)$$

$$\text{(Sum Rule)} \quad P(A + B) = P(A) + P(B) - P(A, B). \quad (2)$$

## Probability Distributions

Much of what follows relies on the concept of a probability distribution function, or PDF. We will use this terminology for both discrete and continuous variables for simplicity, understanding that for a continuous variable,  $P(x)$  means  $P(X$  is between  $x$  and  $x+dx$ ). PDFs sum (or integrate) to unity over all values that the variable can take. For simplicity, we may write  $N(\mu, \sigma)$  to represent a Gaussian distribution with a mean of  $\mu$  and standard deviation of  $\sigma$ .

## Conditional Probabilities

Conditional probabilities represent a partitioning of the data space based on the value of another variable (or variables). Therefore, if A represents the statement “I bring an umbrella to work” and B represents “it rains that day,” then  $P(A|B)$  and  $P(A|\sim B)$  partition all the days into those with rain and those without.

One of the problems we will encounter is that such partitioning can rapidly reduce the number of samples from which to derive information. Take, for example, a patient sample size of 100, which may be sufficient to estimate the frequency of

values for some variable of interest  $A$ . If, however, we wish to condition on two different variables  $B$  and  $C$ , each of which has four discrete possibilities, then  $P(A|BC)$  must necessarily partition the sample space into 16 possibilities, namely “ $B=b_1$  and  $C=c_1$ ,” “ $B=b_1$  and  $C=c_2$ ,” etc. It is entirely possible that one or more of these groups has no samples at all—making it impossible to empirically estimate  $P(A|BC)$  from that data for those values of  $B$  and  $C$ .

## Information Entropy

Information (or “Shannon”) entropy  $H$  is analogous to thermodynamic entropy [5] in that it is a measure of the disorder in a system, and is derived from the possible ways that a system can be arranged while preserving its macroscopic attributes. In the case of information entropy, the “disorder” measurement can be expressed as the smallest number of bits that a large binary string can be compressed, while maintaining all the information it contains (maximum lossless compression). A string like “11111111111111” could be compressed to “15 ones,” for example, while 011101100110000 is much more difficult to compress and thus has more entropy.

The mathematical definition for entropy is derived from the exponential expansion coefficient  $H$  in the limit equation  $N=e^{nH}$ , where  $n$  is the number of measurements of a random variable, and  $N$  is the number of all possible combinations of measurements of length  $n$  that yield the observed distribution of outcomes, e.g. half zeros and half ones. For a two outcome experiment, for example, Stirling’s approximation to the binomial probability distribution function is just such an

exponential, with  $H$  being a function of the fraction of observed trials of one outcome, as well as the probability of that outcome.

In the limit of large  $n$ , information entropy  $H$  in the discrete case is defined by

$$H(X) \equiv - \sum_x P(x) \log P(x), \quad (3)$$

where the summation is over the allowed values of  $X$ , and  $P(x)$  represents the frequency in which that value appears in the system [6]. The base of the logarithm is arbitrary, but is typically taken to be base 2 by those working in information theory, and the resulting units are called “bits.” Extending the definition to the joint entropy  $H(X,Y)$  yields

$$H(X, Y) \equiv - \sum_{x,y} P(x, y) \log P(x, y). \quad (4)$$

Entropy has a maximum value when the probabilities of all possible values of the variable (or variables) are equal. A proof is in Appendix A: Mathematics. The minimum entropy of zero occurs when a variable always results in a single value, so that  $P(x) = 0$  or 1 and all terms in equation (3) vanish.<sup>3</sup>

Conditional entropy, which is the entropy remaining in one variable given the state of another, is written

$$H(Y|X) \equiv - \sum_{x,y} P(x, y) \log P(y|x). \quad (5)$$

---

<sup>3</sup>The value of  $0 \log(0)$  is 0, as can be shown by taking the limit of  $x \log(x)$  as  $x$  goes to 0.

## Mutual Information and Conditional Mutual Information

One novel element of the work described here is that, rather than use more traditional tests for the correlation of variables, we use the information theory concept of mutual information, or MI. Mutual information is a measure of the information gained about one variable when another is known.

MI is strictly non-negative, and does not depend on any specific type of correlation, as would a linear correlation coefficient. The data we will use has no known underlying natural distribution (such as Gaussian), and many traditional statistical tests may fail in this environment. We will therefore find it necessary to empirically model the distributions based on “training data.” The ability to use MI as a model-free test for independence is therefore crucial, since it does not require assumptions about the underlying distributions.

Mutual information has been used since shortly after the introduction of entropy by Claude Shannon [5] in the middle of the last century<sup>4</sup> and has clear probabilistic meaning. Its model-free nature and ease of computation with empirical data make it a natural method for discovering associations between variables.

Mutual information is defined by

$$MI(X; Y) \equiv \sum_{x,y} P(x, y) \log_2 \frac{P(x, y)}{P(x)P(y)} \quad (6)$$

---

<sup>4</sup> Shannon’s classic 1948 paper defines all the terms in the entropy form of the mutual information equation below, but did not explicitly address the concept of mutual information. This paper led to the pseudonym “Shannon Entropy.”

where the sum is over all possible values of the variables  $X$  and  $Y$ . In the case of a continuous variable, the summation is replaced by a double integral over  $dx$  and  $dy$ .

In terms of entropy, the mutual information is

$$\begin{aligned} MI(X; Y) &= H(X) + H(Y) - H(X, Y) \\ &= H(Y) - H(Y|X), \end{aligned} \tag{7}$$

as can be shown by expanding the logarithm function in equation (6) above and applying the definitions of entropy. The minimum value of MI is zero and occurs when  $X$  and  $Y$  are independent variables. In that case,  $P(X, Y) = P(X) \cdot P(Y)$ ,<sup>5</sup> the logarithm vanishes for all terms in equation (6), and the MI equals zero. This can also be seen by examining the joint entropy  $H(X, Y)$ ; in the case where  $X$  and  $Y$  are independent, equation (7) becomes

$$\begin{aligned} H(X, Y) &= - \sum_{x,y} P(x)P(y) \log_2 P(x)P(y) \\ &= - \left[ \sum_{x,y} P(x)P(y) \log_2 P(x) + \sum_{x,y} P(x)P(y) \log_2 P(y) \right] \\ &= - \left[ \sum_x P(x) \log_2 P(x) + \sum_y P(y) \log_2 P(y) \right] \\ &= H(X) + H(Y). \end{aligned} \tag{8}$$

Here the second step relies on the property of products inside the logarithm, the third step on the fact that the sum of  $P(X=x)$  across all  $x$  is one, and the final step applies the definition of entropy. Substituting this result into the first line of equation (7) shows that the MI vanishes for independent variables.

---

<sup>5</sup> This is the definition of independence between two variables.

The maximum value of MI occurs when the result of sampling X always determines the result of sampling Y (this assumes X has the same, or more, possible values than does Y; if not, swap the variables). This maximum value is equal to the entropy of the variable with fewer possible values, and, at a maximum state of entropy, is the logarithm of the number of those values. See Appendix A: Mathematics for a proof.

Another useful way to think about mutual information is as a decrease in information entropy between that of two sets of outcomes taken separately, and the set of outcomes taken together, as can be seen in the first line of equation (7).

Conditional Mutual Information (CMI) is the mutual information between two variables when conditioned on a third. Data is grouped using each of the possible values of the conditioning variable, and the mutual information is calculated between the other two. Explicitly,

$$MI(X; Y|Z) = \sum_{x,y,z} P(x, y, z) \log_2 \frac{P(x, y|z)}{P(x|z)P(y|z)}. \quad (9)$$

## Bayes' Theorem

Bayes' Theorem was used extensively in the development of the classifier and associated algorithms described here. The key feature of Bayes' Theorem is that it allows one to invert the statements inside a conditional probability, e.g. to go from  $P(A|B)$  to  $P(B|A)$ . This is an important step in many analyses, and one that is often not well addressed—especially in traditional statistics. A Student's t-test, for example, answers the question “what is the chance that we would observe these

two sets of data, given that they came from the same underlying distribution?” The real question often being posed, however, is “what is the chance there was a single underlying distribution, given these two sets of observed data?” This latter question is answered by applying the t-test, then using Bayes’ Theorem to invert the resulting  $P(\text{data} | \text{distribution})$  into the required  $P(\text{distribution} | \text{data})$ .

In our experiment, we examine groups of patient samples of known disease state to empirically estimate the distribution for “the probability that we would get this set of data from a serum sample, given that a patient has this disease, and the models we have developed from others with the disease.” The question we really want to answer (for a classifier) is, of course, “what is the probability that a patient has a disease given this set of data derived from their blood serum and the model we have developed from previous cases?” Bayes’ Theorem allows us to make this logical transition. In its most common form, it is

$$\text{(Bayes' Theorem)} \quad P(B|A) = \frac{P(A|B)P(B)}{P(A)} \quad (10)$$

which is easily derived by using the product rule (1) twice—swapping A and B—and solving for the form above. The term in the denominator,  $P(A)$ , is a normalization constant which can be calculated by marginalization (summing over all possible values) of the joint distribution,

$$\text{(Marginalization)} \quad P(A) = \sum_{\text{all } B} P(A, B) \quad (11)$$



which is equivalent to summing terms like the numerator in equation (9) over all possible values of B instead of a particular one.

The term  $P(B)$  in the numerator is called a *prior*. It must be assigned by the researcher as the probability of B before anything is known about A; in the case of the disease example above, it would be the original probability that a sample comes from a patient with the target disease. This would be the researcher's best estimate based on the origin of the sample (e.g. from the general population, or someone with symptoms) but without consideration of the current data. As an illustration of the importance of this term, consider the following (oft-misunderstood) example:

As a requirement for employment, you are required to be tested for a rare (one in a million) but deadly disease. The test for this disease has a *false positive* (see Glossary) rate of 1%, and a *false negative* rate of 1% as well. You test positive for the disease. Bayes' Theorem should give you some relief; since, out of 100 million people tested, it is far more likely that you are one of the million people that test positive falsely, than one of the 99 out of 100 million that have the disease and test positive correctly. It is the prior probability that you have the disease—one in a million—that creates this counter-intuitive result.

To illustrate this mathematically, take the ratio of the probability you have the disease to the probability you do not have the disease, given that you have gotten a

positive test for it. Those values are  $P(B=\text{“have disease,” given } A=\text{“test positive”})$  to  $P(\sim B=\text{“no disease,” given } A=\text{“test positive”})$ .

Applying Bayes' Theorem (10) to both of the terms in the ratio, then cancelling the normalization factor  $P(A)$  that appears in both, yields the ratio

$$\frac{P(B|A)}{P(\sim B|A)} = \frac{P(A|B)P(B)}{P(A|\sim B)P(\sim B)} = \frac{(.99)(10^{-6})}{(.01)(.999999)} \cong \frac{1}{10,000}.$$

Therefore, you are 10,000 times more likely to have gotten a false positive than to have the disease.

These mathematical tools will be used in the development of the algorithms described later. First, however, it is important to understand the data for which those algorithms are designed.

## **CHAPTER 3: DATA SETS AND SIGNAL PROCESSING**

One of the first steps in the process of biomarker discovery is the collection of the biologic samples that will provide the data. The two real data sets discussed in this work originated as blood samples taken from patients diagnosed both with, and without, a specific disease. A third data set, consisting of data computationally generated to mimic the known qualities of the real data, serves as a quality control and testing experiment.

The samples were prepared by technicians at Eastern Virginia Medical School in Norfolk, Virginia. Specifics of the sample preparation appear later in the text, however, the basic process includes:

- Sample collection and labeling
- Sample preparation, including randomization
- Mass spectrometry
- Signal processing

Following the collection of the raw machine data, the signal processing, which is described in detail on page 28, was accomplished. After those two steps, the classification and feature selection methods described in Chapter 4 were applied.

### **Sample Collection and Bias Avoidance**

As has been extensively discussed in the literature recently, the collection and preparation processes, if not done correctly, can introduce biases that make accurate data analysis difficult or even impossible. Baggerly [7] argued that a study claiming to have discovered a biomarker for ovarian cancer was fatally flawed

because samples were ordered according to disease state during measurement. Time-dependant instrument errors then introduced artifacts in the resulting data that made any conclusions inherently suspect.

Even as simple of an error as taking samples from different disease groups at different times of the day could effectively ruin a study, since protein expression may be dynamic, as was noted previously. Sample collection is by far the most difficult and costly part of the process, and those doing the data analysis typically have no control over this phase.

Similarly, the creation of data from the samples must be as unbiased as possible. Data from the two real sample sets described in this work were created by Semmes, et al., of Eastern Virginia Medical School (EVMS) in Norfolk, Virginia [8]. Special care was taken by that group to randomize the processing of the samples to avoid the problems previously described. Also, intermixed with the primary samples were samples taken from a single serum pool (mixture) of a large group of nominally healthy people, which acted as a surrogate for a population average.

This “Quality Control,” or QC, pool allows the measurements to be calibrated in several ways. Since the QC samples are nominally identical, any variations noted must arise from the data creation process of preparation and MS measurement. We have noted (and corrected for) variations due to the position of the sample on

the plate, or “chip,” that is inserted into the machine, and the number of samples run since the beginning of the experiment.

## Leukemia Data

The samples producing the first real data set were provided by the National Institute of Health to Eastern Virginia Medical School, and kept frozen until processed through a SELDI instrument in 2004 [8]. Patients were diagnosed at the time of the original specimen collection by World Health Organization guidelines as to whether or not they had leukemia, a cancer of the blood.

The working data set includes 145 different patients, of which 78 were classified during the clinical portion as “normal,” and 67 with various stages or forms of leukemia.<sup>6</sup> The samples from the patients were processed multiple times, resulting in 425 cases for the study. Multiple cases from the same sample are called *replicates*. Figure 2 is a heat map of the Leukemia data set. Each row of pixels represents the abundances for all the molecules (or peaks) found in a single spectrum (or case); each column is the abundance of a specific molecule for all cases. The color of the  $(i,j)$  pixel reflects the abundance of peak  $i$  in case  $j$ , where  $i$  runs along the horizontal axis. When sorted into classes, the heat map may be useful for searching for diagnostic portions of the spectra by eye. The dotted line represents the division between the normal class, which is the top half of the cases, and the

---

<sup>6</sup> We included acute and chronic forms of lymphoma and myelogenous leukemia, as well as adult T-cell leukemia and other non-Hodgkin’s lymphoma. There are a number of other patients included in the data set with other stages or types of leukemia, however, these subsets were not examined for this study.

disease class. It should be possible to see the class difference in the rightmost variables (disease cases are slightly brighter) that we will later find to be diagnostic.

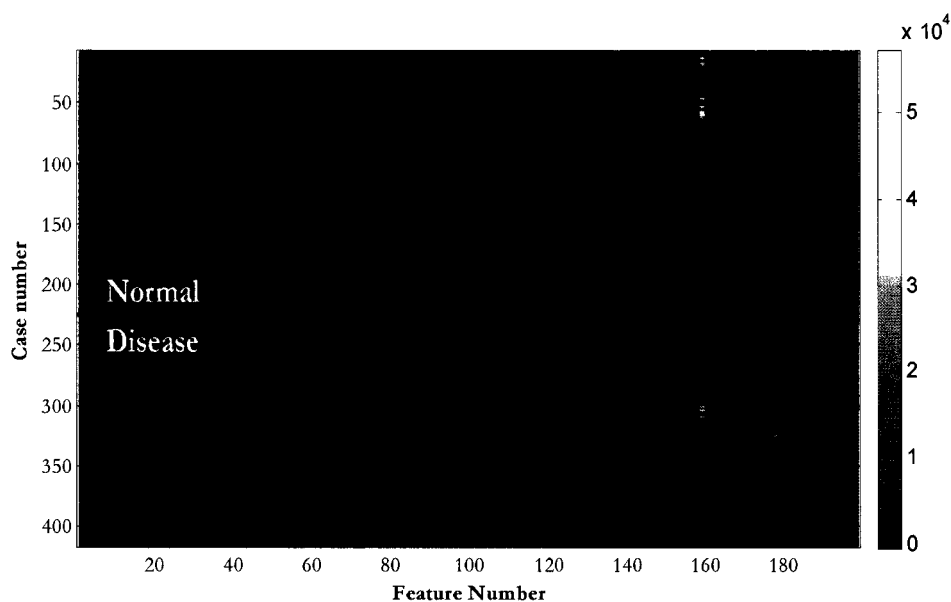


Figure 2: Leukemia data

One of the samples in the “leukemia” class was rejected after it was found that the database had a transcription error. That sample’s original diagnosis was not compatible with the “leukemia” classification. Another sample, diagnosed as “smoldering leukemia,” was rejected since that condition is considered a “preleukemia” [9] and only develops into leukemia in a minority of cases [10]. One of the four replicates of another sample was found to differ greatly<sup>7</sup> from the other three, and was also removed. Figure 3 shows the mislabeled replicate 1 in blue, superimposed on the correct replicates (2-4) of the same ID number.

---

<sup>7</sup> We examined linear correlations between replicates of each sample. Those which had low correlation with other available replicates were examined manually. All but one was retained, as the low correlations appeared to be due to signal variations. The one rejected appeared to be completely unrelated to the three other replicates with the same sample ID number.

Therefore, the analysis was done with 65 total cases in the disease class, and 417 total replicates, with  $m/z$  up to 13 kDa.

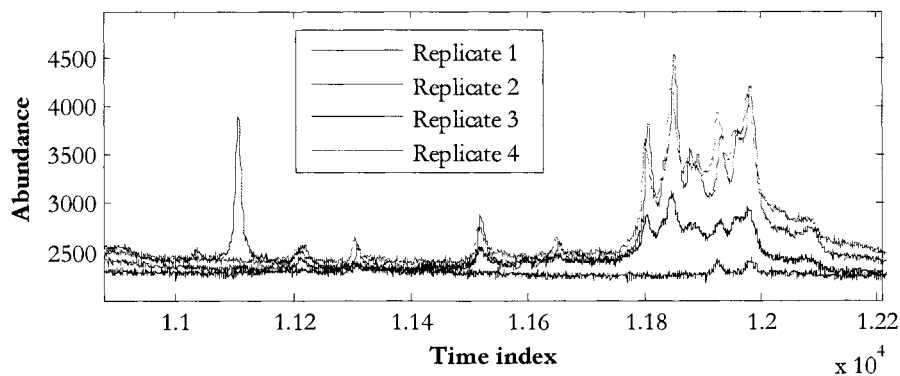


Figure 3: Mislabeled replicate spectra

During the course of the development of this methodology, others in our group continued to work toward more accurate identification of peak positions and their values. Because of this increased fidelity, we had access to four separate versions of the data set. The first had 48 unique  $m/z$  positions, the second had 120, the third 199, and the final set had 209 unique variables identified. The final version is the one primarily referenced in this work, however, in Chapter 5: Application of the Naïve Bayesian Classifier, the first version was used for testing.

## Generated Data

The computationally generated data set strives to mimic as closely as possible the 199-variable version of the Leukemia data set. The numbers of cases, including replicates, and number of peaks are similar.

In this data set, we attempt to reproduce those systematic and statistical properties we have found in the real data, without the several artifacts that we have no specific explanation for (such as certain peaks failing to appear in some replicates).

The primary purpose of this data set is for quality control and testing of the algorithms. By mimicking known properties of the real data, then attempting to identify those properties with algorithms made for that purpose, we gain a better understanding of the reliability and stability of the protocols used.

The following steps were taken to prepare the generated data:

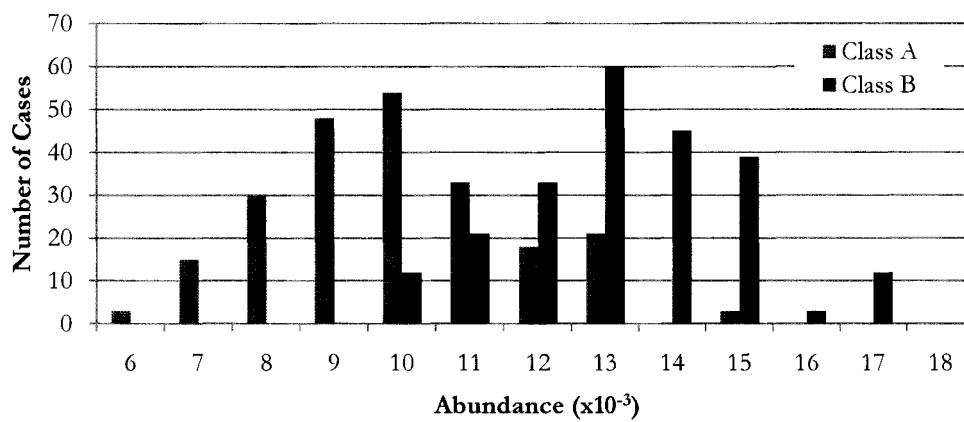
1. A spectrum<sup>8</sup> with 200 peaks is created by taking the average of the non-disease cases in the Leukemia data set. This provides a baseline for creating all the cases that will be used.
2. A set of spectra, with the number of cases approximating the number of unique patient identification numbers in the Leukemia data, is generated via a draw from a  $N(\mu, \sigma)$  distribution for each variable independently.  $\mu$  is the value of the average spectrum at that peak position,  $\sigma$  is estimated from the Leukemia data set population. At this point there should be no real distinction between any of the 200 variables.
3. One-half of the population is designated to be in the disease class. A class vector representing this choice is created and attached to the data.

---

<sup>8</sup> A full spectrum is not created as we do not wish to replicate the signal processing steps described later. Instead, the steps here are applied to the final peak list data.



4. One peak (labeled 200) is chosen as “highly diagnostic” and the mean values of the two subpopulations (normal and disease) are separated by two times the population’s average standard deviation. Specifically, the disease cases are redrawn from  $N(\mu+2\sigma, \sigma)$ . This results in a distribution like the one shown in Figure 4.



**Figure 4: Generated data distribution for highly diagnostic peak**

5. A random fraction (about a tenth) of the total value of this peak is placed into each of four adjacent peaks (labeled 195-199). In this manner, five diagnostic peaks are created, all diagnostic of the class. This procedure mimics the measurement of adducts or modifications in the real data set, wherein slightly modified molecules show up as peaks separate from the original.
6. A small fraction of the value of the key peak (200) is moved into a peak some distance away in the list (labeled 100), representing a multiply-

charged ionization satellite ( $z = 2$ ). This is repeated to a different peak (labeled 99) for one of the adducts (199).

7. Another moderately diagnostic<sup>9</sup> peak is created but not added to the peak list. Instead, varying portions of the total value of that peak are placed in two non-adjacent peaks (labeled 50 and 150). This represents the breaking apart of a biomarker protein, whose mass is too great to be detected, into several fragment molecules that are in the range of measurement.
8. Two more peaks (labeled 1 and 2) are selected as “mildly diagnostic” and the values chosen from two normal distributions whose means are separated by about one standard deviation of either group. Specifically, the disease cases are redrawn from  $N(\mu + \sigma, \sigma)$ . One of these two peaks has a portion of the other peak’s value added to it to represent two peaks that are so close together that the peak value of one is “riding up” on the tail of another. See Figure 8 on page 30 for an example.
9. The cases are replicated three times (the original of each case is discarded) by multiplying each value by a de-normalization factor to replicate the signal strength and chemical preparation effects as described on page 31. For a single data vector  $\mathbf{X}$ , a factor  $f$  is first selected from  $\sim U(0.5, 2.0)$  to replicate the range of total ion current normalization factors found in the

---

<sup>9</sup> Difference in means is about one and a half standard deviations of the sub populations.

Leukemia data. The resulting distribution for the highly diagnostic peak is shown in Figure 5.

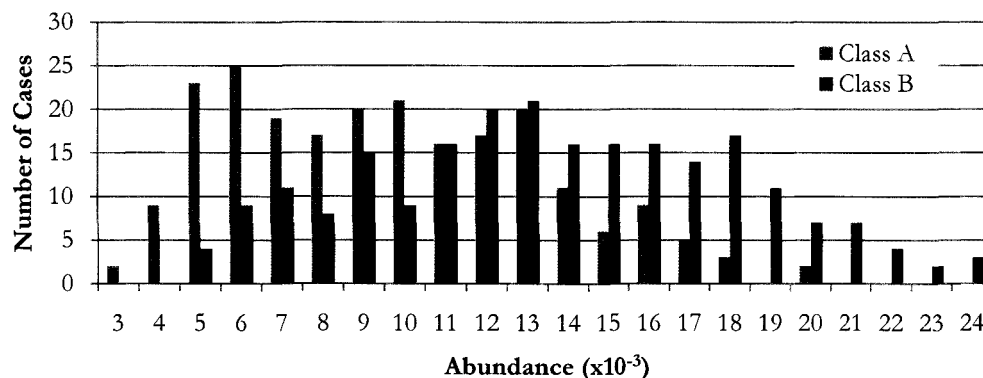


Figure 5: Distribution for highly diagnostic peak after de-normalization

A summary of the diagnostic peaks placed in the generated data is given in Table 1. The resulting Bayesian network is shown in Figure 39: *Resulting Bayesian network, generated data*, on page 107 .

Table 1: Diagnostic variables, generated data

Peak	Purpose
200	Highly diagnostic
196-199	Adducts or modifications of peak 200
99, 100	Correlated doubly charged ionization states of 199, 200
1, 2	Diagnostic with correlations due to mixing
3, 4	Mildly diagnostic
50, 150	Diagnostic—but hidden—primary peak

The actual data set is too large to include in this document, but a heat map is shown in Figure 6 below. The code for creating the generated data can be found in Appendix B: MATLAB Code.

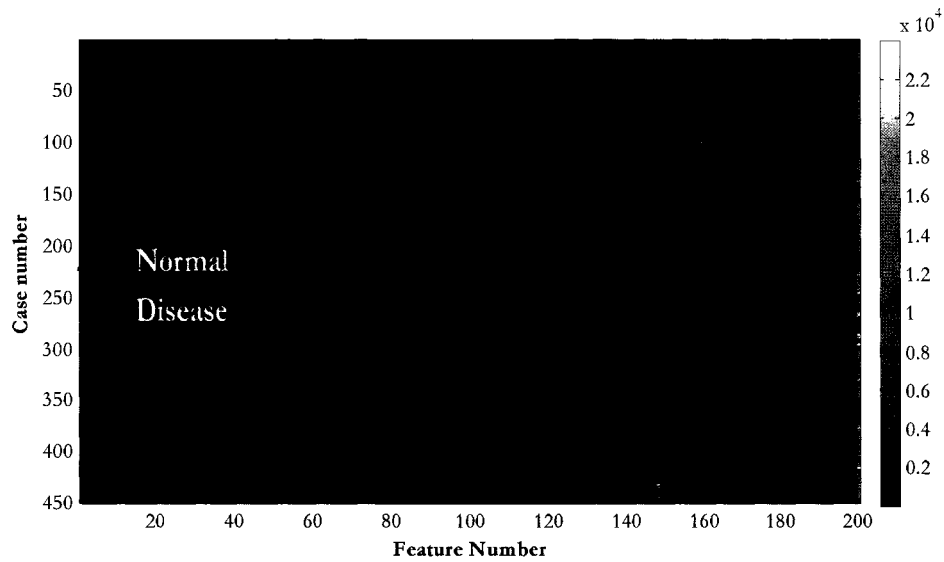


Figure 6: Generated data

## Prostate Cancer Data

The data encompassing the prostate cancer (PCA) data set was created under our ongoing National Cancer Institute-funded project. The data is secondary to the primary goal of that project, which is “improved signal processing methods” of the type described in the following section.

Serum samples selected for this study were chosen to have a wide range of prostate-specific antigen (PSA) levels, with similar PSA distributions in both class groups—disease and normal. Previous studies have had high PSA levels in only the disease group; we wished to avoid the possibility of introducing experimental bias.

Therefore, on this project, samples were selected to be included in the non-disease group based on having PSA levels that matched those of samples found in the disease group.

As in the Leukemia data set, several replicates were created from each serum sample. These replicates received independent chemical preparation. Two affinity surfaces (IMAC and C3) were used for protein purification from serum. The affinity surfaces assist in enhancing the signals for certain types of molecules such as the hydrophobic apolipoproteins (C3) or phosphorylated proteins (IMAC) [11].

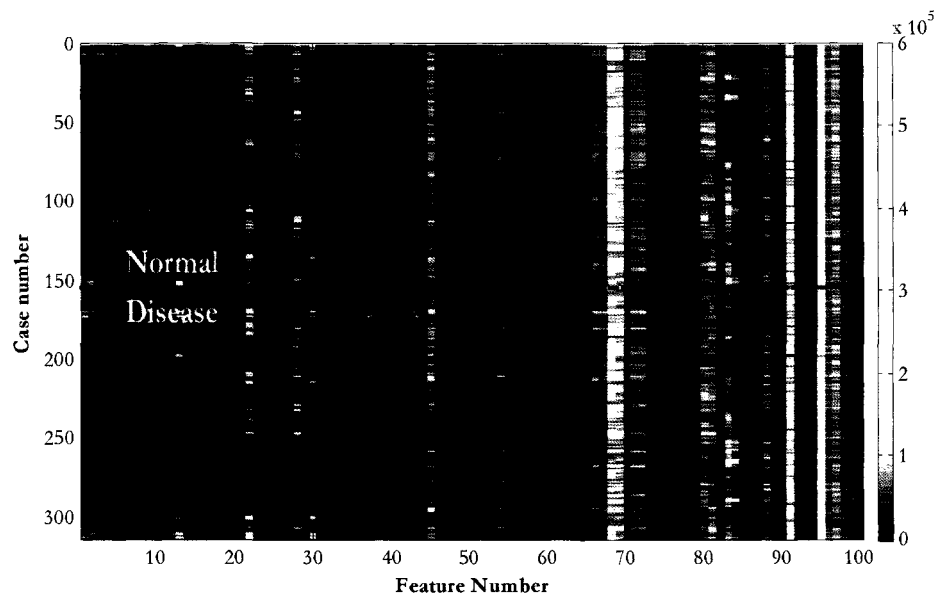


Figure 7: PCA data

The Bruker Ultraflex instrument used required the spectra to be gathered in three stages for maximal resolution. Spectra were taken in the mass ranges of 0-20kDa, 15-100 kDa, and 2-100 kDa; here we use the results from the 2-100 kDa mass range. More detailed information on the exact experimental design and MS equipment can be found in Gatlin-Bunai (2007) [11]. A heat map of the PCA data is presented in Figure 7.

## Signal Processing

For the leukemia and PCA data sets, sample order was randomized and patient samples were interspersed with QC samples. The MS measurements were run over a period of several weeks. For each replicate of a sample, a spectrum was produced and tagged with metadata, including patient ID, date collected, and date/time and settings of the MS run, among others.

These several replicate spectra from each sample, along with the metadata described above, constitute the input to data analysis portion of the project.

## Data Analysis

There are three phases in the data analysis process – signal processing, feature selection, and classifier construction. The first, whose input is the set of spectra and associated metadata, includes a number of steps, listed in Table 2. This phase is not the primary concern of this work, but it is necessary to understand the steps in this phase, and especially the problems caused by their imperfections, to understand the results of the final two phases.

The output of the signal processing phase is a two dimensional table of values, in which each row represents a single replicate spectrum, and each column represents a mass per unit charge ( $m/z$ ) spectral position. The table entries are the measured signal intensity of that replicate at that  $m/z$  value.

**Table 2: Steps in the Data Creation Process**

<b>Background Subtraction</b>	The presence of large amounts of matrix molecules and other instrument effects produce a slowly varying underlying signal that must be removed so that a true zero value can be found and peak heights measured against this baseline. See Figure 8.
<b>Resampling</b>	Because peaks at the higher mass end of the spectrum have broader width than those at the lower end, our group resamples all peaks to have the same width (in the time domain). Therefore, a peak that is 10 time units wide may be resampled to be a standard of 5 time units wide, and its height doubled, to maintain the total integrated signal under the peak. This step allows for more accurate peak selection and alignment. It also increases the accuracy of our normalization procedure, discussed later.
<b>Peak Selection</b>	Peak selection determines which $m/z$ values represent a molecule, and which of those appear in a sufficient number of samples to represent a variable in the end data. The noise associated with the instrument can mask peaks of lower signal intensity, or conversely, appear as peaks where none exist.
<b>Peak Alignment</b>	Ensures the same true $m/z$ value (representing a specific molecule) is in the same position in the data table for all samples. See Figure 8.
<b>Recalibration</b>	Corrects for the effects of instrument errors. The QC data is examined for changes in total signal over time, for example; this calibration is then reapplied to the sample data.
<b>Replicate Averaging</b>	To reduce variations due to preparation and measurement, several samples are measured from a single patient's serum. The results are initially treated as independent samples for peak selection, alignment, and other steps, but are eventually averaged to give a single measurement for each patient.
<b>Deconvolution</b>	A peak whose mean value lies within one peak width of another, which is often the case with adducts, rides up on the slope of the adjacent peak. The adjacent peak must be deconvolved (removed) to find the true maximum value of the neighbor. See Figure 8.

Figure 8 shows the necessity for several of the data creation steps listed above. The first pane shows the extreme background height arising at lower  $m/z$  values. This

background can cause dependencies in the values of nearby peaks (peak A high implies peak B high) even when no real dependency exists, since the level of the background overshadows the true level of the peaks. Removal of this “background” signal reduces artificial correlations between variables and the variance between samples.

The second pane shows a slight shift to the left between the peak maxima for two spectra, even though these spectra are derived from the same pool of sera. This shift can cause incorrect measurements of either the peak position or the peak value, or both, and may even cause peak detection to fail. Peak alignment attempts to find time-scale correction coefficients to ensure peaks from similar ions appear at the same mass positions in all spectra, and are measured at their maxima.

The third pane shows that when peaks have  $m/z$  value differences less than the peak width, portions of the peaks can add up to yield a value that is higher than the true abundance for either peak. This will also yield artificial correlations between peaks (massive left peak always implies high right peak). Deconvolution attempts to find the true maximum values of closely spaced peaks.

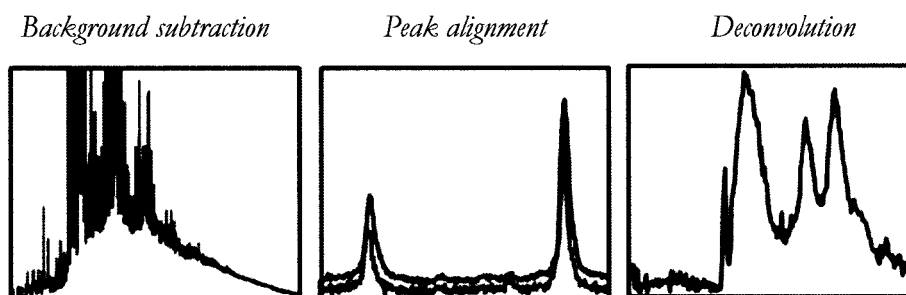


Figure 8: The need for signal processing



Once all of these steps are completed, the abundance values at the aligned peak positions are recorded, along with other data about the spectra. Table 3 shows an example of the output of this process.

**Table 3: Example abundance values for five patients (arbitrary units)**

<i>Patient ID</i> <i>number</i>	<i>Disease</i> <i>Class</i>	<i>m/z positions</i>			
		2755	2797	2873	2959
<b>665</b>	Normal	1.368	0.308	2.151	0.774
<b>672</b>	Normal	1.600	1.827	1.798	1.636
<b>679</b>	Normal	0.399	1.630	1.749	1.418
<b>696</b>	Disease	0.438	0.696	1.607	1.941
<b>721</b>	Disease	1.249	1.023	1.944	1.106

### Normalization

One critical problem with of an MS experiment is that two identical samples (or even two scans of a single sample) can result in spectra that, while having nominally the same shape, differ greatly in the values of the various peaks. As noted previously, sample preparation, particularly total volume of material, plays a large role in producing this systematic error—but other factors such as ionization efficiency add to the problem.

Several methods of correcting for such errors have been used [12], however, we have determined that a simple method of total ion current normalization reduces much of the sample-to-sample variation without undue complexity that might introduce more artifacts. We have noted unexpectedly high correlations between some variables in the QC data due to problems in signal processing, and normalization does reduce, but not eliminate, these correlations [13].

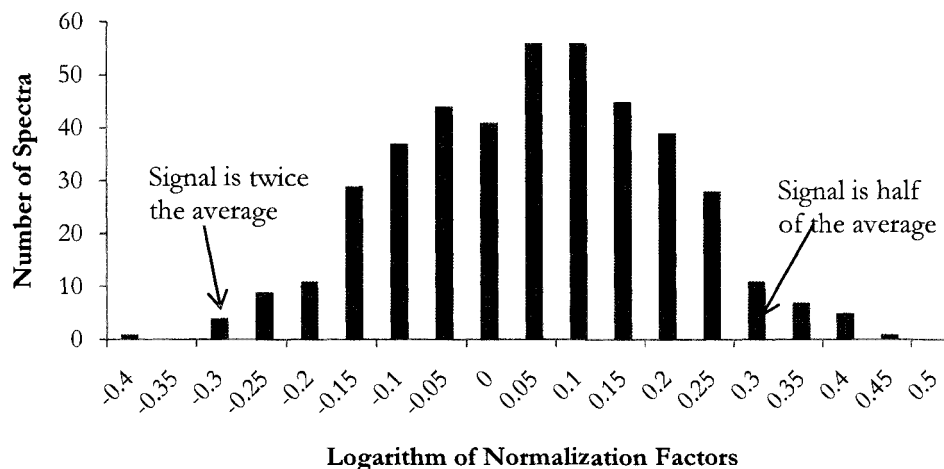
For our method of normalization, the signal processing described in Table 2 is performed, creating an array of signal intensity measurements. Each sample (row)

is summed across all peak positions (columns) to find that sample's *total ion count*. An alternate method, integrating the processed spectra across the entire  $m/z$  range, was considered but discarded due to its higher dependence on precise background subtraction—a process with relatively large inaccuracies.

Every abundance value in each sample is then scaled by a normalization factor equal to the population average total ion count divided by the sample total ion count. This method reduces the variation in measured abundances of nominally identical samples, such as those from the QC data, from 40-50% to about 20% of the average value.

It is possible to normalize the sample total ion count on subsets of peaks. We avoid this technique, however, due to the possibility of destroying valid information should we “normalize out” variations in samples from different classes. While this is a possible problem with total ion normalization as well, we feel that the chance of introducing error is greater with a small subset of peaks used without additional information.

Figure 9 shows the normalization factors resulting from the process described above being applied to the Leukemia data set of 417 spectra. The horizontal axis lists the possible values of the normalization factor  $\varphi$ , shown as  $\log_{10}(\varphi)$ . A factor  $\varphi = 1$  (no normalization required) is represented by the zero position. Positive 0.3 represents spectra that were doubled to bring the total ion count to the data set average; negative 0.3 represents spectra whose signal was halved.



**Figure 9: Histogram of normalization factors**

Inspection of the spectra with normalization factors above 2 show that these spectra have consistently low signals and, therefore, lower signal-to-noise ratios. This induces data reduction errors, especially in background subtraction and peak picking. It is also clear, by inspection, that these low signals are not necessarily an attribute of the sample, as some low signal spectra have replicates (other spectra produced from the same sample) that have no apparent problems.

We have therefore chosen to include in our algorithms the option to remove spectra with high (greater than 2.0) normalization factors. While this choice of threshold was somewhat subjective, in the chart above it is apparent that there is a large decrease in the frequency of occurrence at about that value. In no case were all the various replicates of any one sample removed.

The resulting array of signal intensities for each replicate at each peak position, as well as the metadata necessary to identify each spectrum and its class, constitutes the input to the final phases, which are the primary focus of this work.

## CHAPTER 4: CLASSIFICATION AND FEATURE SELECTION

Each  $m/z$  column in our data arrays, which started out as a signal peak found in a number of spectra, can be considered a set of realizations of a random variable representing the measurement of the abundance of that molecule in each sample. Each row, created from a single spectrum, is an instance, or case, of the full set of variables.

These variables are also referred to as “features.” The terminology arises from information theory and the computational task of pattern recognition. In this task, an image is broken into features—such as the eyes and nose of a face—that are significant, and only specifically selected features are processed. We will transition to this terminology for the remainder of the discussion, understanding that the terms “peak,” “variable,” and “feature” are synonymous.

The phase of our analysis that follows the signal processing is called “feature selection.” In this phase, features are chosen for inclusion or exclusion in the classifier, the goal being that the final feature set includes only those variables that will be helpful in classification.

The final phase is the construction of a classifier. This construct, whose parameters are typically learned from samples of known classification, allows new, unknown samples to be classified—in this case for disease state. A probabilistic classifier returns the probability that the sample lies in one or another class (e.g.

“95% chance of having Leukemia”). A deterministic classifier takes the values of the features and returns a specific classification. A popular choice, and the one we have made, is to build a probabilistic classifier and use its output to give a deterministic result by setting some probability threshold for declaring the class, such as “if the probability that this case comes from a patient with leukemia is greater than 50%, we will consider the class as “disease.” Obviously, this may not be the threshold that a doctor might set for further testing of a patient. It is common to vary this threshold to understand the relationship between the false positives and false negatives that result; we have not done so here, as the accuracy of the classifier is not our primary goal.

Instead, the primary focus of biomarker discovery lies in the feature selection phase. Initial efforts in this field focused on finding single features that are indicative of disease state, although more recently, multi-feature sets (or even mathematical combinations of features) have been sought and examined (cf. Oh, 2005 [14]). Once diagnostic features are found, further investigation as to the nature and origin of the molecules is done in an attempt to learn more about the processes causing the disease itself. Diagnostic features are not typically called biomarkers until their underlying biology and relation to the disease is better understood. In this research, we limit the process to selection of diagnostic features, although we will attempt to examine their inter-relationships.

## Feature Set Selection

As has been emphasized, the choice of which features to include in a classifier is the primary goal of this research. These features represent  $m/z$  values, which may lead to the identification of proteins, then genes, and then the biological processes that may cause the disease. Even if we could create an accurate classifier directly using all the variables produced by the data reduction methods discussed in Chapter 3, we would still want to identify those features that provide the greatest information.

### Filter and Wrapper techniques

Choosing which features to include in a specific classifier can be accomplished using a number of criteria. Methods that use some scoring criteria to select individual features prior to creating a model for the classifier are known as *filters*.

Another commonly used method is that of the *wrapper*, which selects a feature subset and scores it using the resulting classifier itself, say by the classifier's error rate for a data set with known results. An algorithm searches through the space of all subsets, looking for ever lower error rates.

The search algorithm used in the wrapper technique is typically not exhaustive. For most problems, the number of possible subsets is intractably large. Instead, one of many approximate search methods is used, eliminating large portions of the search space at each iteration. A good review of the many methods is found in Miller [15].

We have employed both filter and wrapper techniques. Although we have said that the final phase of the analysis is the classifier construction, it is clear that a wrapper technique requires both feature selection and classifier construction to occur simultaneously.

### **Wrapper methods**

Two of the most straightforward wrapper methods are *forward selection* and *backward elimination* [15]. To perform forward selection, the following pseudo-code is implemented:

```
Select a feature
  Find an n-fold cross-validated error rate based on using only
  that feature as a criterion.
Repeat for all features.
Permanently select the feature that has the lowest error rate.
From the remaining features, select a feature.
  Using the new feature, and the one chosen previously, create a
  model and find the model's cross-validated error rate.
Repeat for all features.
```

Choose the feature that, when combined with the first feature selected, results in the best model. Continue to add more features until some threshold—perhaps “error rate begins to rise”—is reached. The final set becomes the selected subset of features.

Backward elimination is a similar process. Under this method, each feature of the current set is removed, one at a time, and the remaining feature set is used to find a cross-validated error rate. The feature whose removal results in the lowest error rate is discarded permanently. The process is repeated until some threshold (error rate rises, or minimum number of features remain) is met. Our first attempt at

feature selection and classification is a wrapper method that uses exactly these techniques.

## Naïve Bayesian Classifiers

Bayes' Theorem, which is a quarter-millennium old, has come into more widespread use in the last quarter century—especially in the fields of machine learning and pattern recognition. One of its most popular uses is for email filtering, where it is used to answer the question “what is the probability this email is spam, given the words and other data in the message” by learning the probability of those features first from a corpus of source-known examples.<sup>10</sup> These probabilities are often updated as the recipient manually accepts or rejects new messages, refining and personalizing the filter.

This “learning” of the probabilities for data occurrence based on a set of cases with known results is a key step in the construction of any Bayesian classification system. The known data is called the “training set,” and the probability that a new case will be observed to have a certain value (or range of values) is based on the frequency of that value in the training set, or a model distribution based on the data in the training set.

As an illustration of the use of learned probabilities and Bayes' Theorem to classify, consider the following example.

---

<sup>10</sup> See, for example, the 1997 Microsoft Research group article entitled “A Bayesian approach to filtering junk email” at <http://research.microsoft.com/~horvitz/junkfilter.htm>.



A researcher is searching for the probability that “a person has long hair, given that they are male” using Bayes’ Theorem. To do so, the researcher might sample a population, noting the sex and hair length of each case, then choose some boundary between long and short hair, such as “touches the collar.” A probability distribution is created by directly counting the samples with long (or short) hair. The researcher then makes the assumption that “if I have seen that 80% of the males so far have short hair, then the probability of a new person having short hair, given they are male, will be 80% as well.” This, plus a researcher-chosen prior probability of a new person being male (and the normalization factor) would allow her to answer the question “what is the chance that this new person is a male, given that I know only that they have short hair?”

This example shows the simplest use of Bayes’ Theorem for classification, based on a single variable (hair length). Suppose however that the researcher knows that she will have the additional information that “the new person wears pants (or not).” By gathering this information from the training set, and creating the needed four entry probability table  $P(\text{sex}=\{\text{male, female}\} | \text{pants}=\{\text{yes, no}\})$  she can use this additional data to refine the classifier. If she assumes that the length of a person’s hair and the wearing of pants are independent, i.e. men with short hair are not more likely to wear pants than men with long hair, then by the definition of independence  $P(\text{hair, pants} | \text{sex})=P(\text{hair} | \text{sex})P(\text{pants} | \text{sex})$ .

Using H, P, and S to represent hair, pants, and sex respectively, she writes Bayes' Theorem as

$$P(S|H, P) = \frac{P(H, P|S)P(S)}{P(H, P)} = \frac{P(H|S)P(P|S)P(S)}{P(H, P)} \quad (12)$$

The prior, P(S), is chosen by the researcher based on her opinion of the chance the next person will be male (without regard to any data collected on the next person). The other two terms in the numerator, P(H|S) and P(P|S), are learned from the training set as described previously. As before, the normalization factor in the denominator is calculated by summing terms like the numerator for all values of S.

This two-variable classifier is directly extensible to any number of variables, as long as the independence condition holds. That is, for some set of variables  $\mathbf{X} = \{X_1, X_2, X_3, \dots, X_n\}$  we must have

$$P(\mathbf{X}|C) = \prod_{i=1}^n P(X_i|C) \quad (13)$$

when conditioned on the class C. By extending the classification to utilize several variables, then simplifying the estimation of their PDFs with the independence assumption, we have created what is called a *naïve Bayesian classifier*, or NBC.

This independence assumption is critical to the functioning of the naïve Bayesian classifier. Although it has been shown [16] that an NBC works well in many cases even when the independence assumption is not valid, artifacts can appear that can cause instability in the classifier.

One problem is that the dependencies impose exaggerated values in the estimation of the final probability. To illustrate this, consider a system of two variables, A and B, and the problem of building a classifier  $P(B|A)$ . Further consider that we attempt to add a third variable,  $A'$ , which is assumed to be independent, but in fact always is observed to have exactly the same value as A through some underlying, or measurement induced, dependence.

By considering the observation as that of a two variable vector  $\mathbf{A}=\{A, A'\}$ , we follow the procedure outlined above to find (ignoring the normalization factor for now)  $P(B|\mathbf{A}) \propto P(A|B)P(A'|B)P(B) = P(A|B)^2P(B)$ , since  $P(A|B) = P(A'|B)$  for all B.

If, for example,  $P(A|B)=0.9$  and  $P(B)=0.5$ , the NBC will determine  $P(B|A)=0.9$  in the case of one variable, but  $P(B|\mathbf{A})=0.99$  in the case of two variables. The true probability is the first, but by measuring the single variable A twice, and erroneously considering the results to be independent, we have greatly overestimated the confidence in the classification.

A second example showing how the NBC can behave poorly due to incorrect assumptions about independence is in Appendix A: Mathematics. We will see, in Chapter 5, that this instability will cause the NBC to be only partially suitable for the experiment we are interested in, since it has many highly correlated variables, such as signal peaks arising from adducts and multiply charged states. These are

indeed multiple representations of a single underlying variable—the abundance of the primary molecule.

## Bayesian Networks

A Bayesian network is, at its most basic level, a formula for a joint probability distribution of a set of variables, such as  $P(A,B,C,D,\dots)$ . This formula can be represented graphically by use of a directed acyclic graph, or DAG.

The DAG has two elements: nodes for each variable in the problem, which we will represent as ovals, and arcs, or lines between nodes. The arcs are directed, so that they point (with an arrow) from one node to the other. The graph is acyclic; there are no nodes where it is possible to start, and then return, by following a set of directed arcs (also called a path). Figure 10 represents a simple DAG with five nodes.

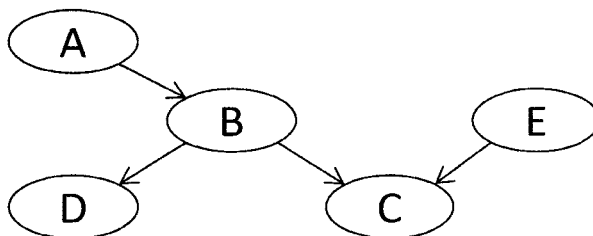


Figure 10: Bayesian network

The DAG encodes a set of facts about the relationships between the variables in the distribution it models [17]. Arcs represent dependencies, so that, in the most basic case of only two nodes, an arc is drawn if they are dependent and no arc if they are independent. The DAG is simply a way to represent all the dependency information in a particular system of variables; mathematical theorems about

various dependencies are then represented as easily visualized operations on the DAG. For more information, see Chapter 2 of Jensen [17].

In the DAG, we call two nodes with an arrow between them a *parent* and *child*. More generally, the set of nodes from which a path can be found to a particular node are its *ancestors*; the set of nodes that can be reached by following any path from a particular node are that node's *descendants*.

Unlike the NBC described earlier, which is a special case of a Bayesian network, there is no specific variable representing class in the general BN—although one may be identified as such if needed. The DAG represents all the variables and all the dependencies at once. In addition, it represents the minimum set of probability terms that can be used to describe the joint probability distribution (JPD) of all the variables.

As an illustration, assume that a particular problem of interest has three variables, A, B, and C. By using the product rule (1), the joint distribution of the conjunction ABC can be written  $P(ABC)=P(A|BC)P(BC)$ ; repeated applications yield

$$P(ABC) = P(A|BC) \cdot P(B|C) \cdot P(C). \quad (14)$$

Assume further, that in this system, we know that A is independent of C given B; then  $P(A|BC)=P(A|B)$ . Rewriting (14),

$$P(ABC) = P(C) \cdot P(B|C) \cdot P(A|B). \quad (15)$$

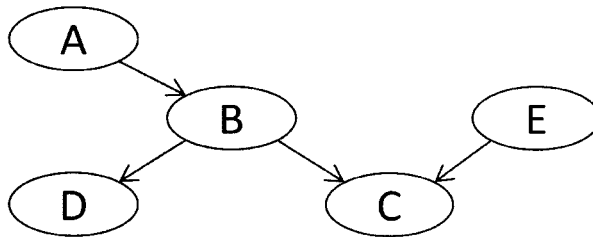
Now consider a DAG of the form  $C \rightarrow B \rightarrow A$ , where the capital letter represents a node in the DAG. If we take a product of terms of the form  $P(\text{node } i|\text{all$

parents of node  $i$ ), then for  $C \rightarrow B \rightarrow A$  we have  $P(C)P(B|C)P(A|B)$ , which is exactly the right hand side of equation (15). In this form, the terms in the product has encoded the independencies we have assumed to exist.

We say that the DAG represents the JPD. The terms resulting from factoring the joint distribution (after applying independencies) can be read graphically from the DAG [17], explicitly

$$P(X_1 X_2 X_3 \dots X_n) = \prod_{i=1}^n P(X_i | \{Pa(X_i)\}) \quad (16)$$

where  $\{Pa(X_i)\}$  represents a conjunction of all the parents of  $X_i$ . Creating the JPD from the DAG is as simple as going through the DAG node by node, writing the terms  $P(\text{node} | \text{parents of the node})$  as a product.



**Figure 10: Bayesian network**

For the DAG in Figure 10, reproduced here, the joint distribution would be

$$P(ABCDE) = P(A) \cdot P(B|A) \cdot P(C|BE) \cdot P(D|B) \cdot P(E). \quad (17)$$

A derivation of the formulae underlying a Bayesian network is beyond the scope of this work, but can be found in Jensen [17]. However, one important consequence of the representation of a JPD by a DAG is that the independencies can be immediately read from the DAG. A serial connection such as  $A \rightarrow B \rightarrow C$

in Figure 10 means that A is independent of C given B. One may think of a node as a valve, if the node is known, the valve closes, and the flow of information is blocked. A diverging, or inverted “V” structure, like  $D \leftarrow B \rightarrow C$ , encodes a similar independence, namely that D is independent of C given B.

To show why this is so, consider the JPD encoded by the simple serial DAG  $A \rightarrow B \rightarrow C$ , which is

$$P(ABC) = P(A) \cdot P(B|A) \cdot P(C|B). \quad (18)$$

By the product rule, we also have

$$P(ABC) = P(AC|B) \cdot P(B). \quad (19)$$

Setting these two equal and solving for  $P(AC|B)$  yields

$$P(AC|B) = \left[ \frac{P(A)P(B|A)}{P(B)} \right] P(C|B). \quad (20)$$

The term in square brackets is  $P(A|B)$  by Bayes’ Theorem. Inserting this yields

$$P(AC|B) = P(A|B) \cdot P(C|B), \quad (21)$$

which is exactly the statement that A and C are independent given B. We say A and C are independent when knowledge of B “breaks the path” between them.

The “V” structure  $B \rightarrow C \leftarrow E$  in Figure 10, however, has an opposite meaning. It represents the statement that B and E are unconditionally independent, but become dependent when C is given (or *instantiated*).

The general JPD for these three variables can be written as

$$P(CBE) = P(C|BE) \cdot P(BE) \quad (22)$$

by the product rule, and

$$P(CBE) = P(C|BE) \cdot P(B) \cdot P(E) \quad (23)$$

by using our method of reading JPD from the DAG.

Setting these two equal immediately yields

$$P(BE) = P(B) \cdot P(E) \quad (24)$$

which demonstrates that the V structure has encoded the independence assumption of equation (13). The proof that instantiation of C “opens up” an information path between E and B is lengthier and can be found in Jensen [17]. We will not have occasion to use the “V” structure in the experiment discussed here.

The “V” structure, the inverted “V” structure, and the serial structure are all the possible combinations in a DAG. It is possible to include an isolated node, which is not connected to any other. Such nodes can be immediately marginalized out of the JPD and do not affect the outcome of a classifier. To illustrate, consider a three-variable network with an arc from A to B, and a variable C not connected to the other two. The JPD is

$$P(ABC) = P(A) \cdot P(B|A) \cdot P(C). \quad (25)$$

Marginalizing (see page 14) across values of C, we have immediately

$$P(AB) = P(A) \cdot P(B|A), \quad (26)$$

which is the JPD represented by just A→B. Thus, the independent variable C is marginalized out of the network with no loss of information—assuming it is not a variable of interest.



## Bayesian Networks and Causality

The most important reason to choose a BN for MS data classification is the BN's ability to reflect causal relationships between variables. As is proven in Pearl [18], if a set of variables have causal relations, and the BN is built such that arcs fully represent the causal paths between the variables, then the resulting BN/JPD will encode dependencies and probabilistic relations between variables.

The reverse statement, that causality can be inferred from a BN that has been created from examination of a particular set of data, is not strictly true [18]. Data can show dependencies where causality does not necessarily exist, or where perhaps an unknown variable linking the two has not been included.

A conceptual causality chain for the problem discussed here is diagrammed below. However, the inclusion of the hidden variables (dotted nodes), such as changes in gene regulation, serves only to provide probability distributions for those unknown processes, and does not affect the outcome of the classifier. They will therefore be suppressed.

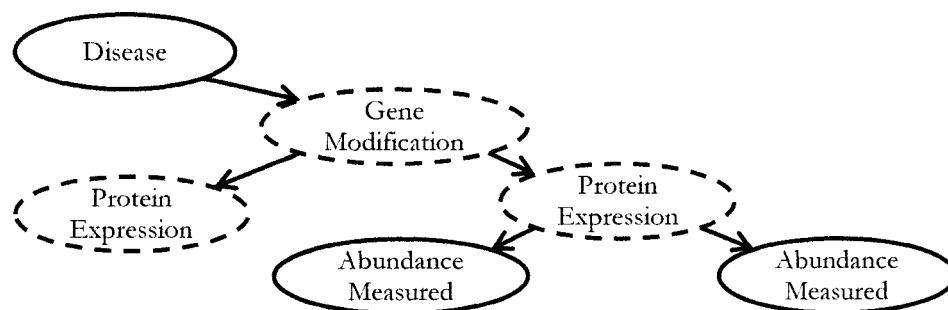


Figure 11: Causal chain for disease

Although not shown here, the variables representing a measurement of the abundance of a particular protein (or fragment) could be connected. We expect, for example, proteins to split into fragments due to the violence of the ionization method with some probability of occurrence. This would be represented by a diverging (inverted V) structure with the primary molecule at the top and the fragments underneath. The fragments can theoretically have a secondary connection to the disease class, should the disease change the probability of fragmentation, but we have not identified such a connection in our work.

A serial connection could also occur, particularly in the case of a multiply-charged ion, or satellite. Recall that the measurement is of the mass-to-charge ratio, so that an extra charge would cause the molecule to show up at  $m/2$  of the primary ion. The variables representing the abundance measurement of these two mass values should be dependant, since high abundance of a parent should imply high abundance of the satellite ion.

Although V structures (representing, for example, two proteins that produce a common fragment) are theoretically possible, we will not seek this structure. Even in the unlikely situation where this occurred, we would not be able to precisely allocate the abundances back to the primary ions without additional experiments.

## **Bayesian Classifier Construction**

There are two primary steps to building a classifier based on a Bayesian Network. The first is to determine the structure of the network. This is the most difficult

part, especially when it must be derived from data which is statistically noisy. The second is to determine the exact values of the terms in the JPD (e.g.  $P(A|BC)$  for all values of A, B, and C). These two steps are called “structure learning” and “parameter learning.”

### Structure Learning

The primary difficulty with learning the best<sup>11</sup> structure of a BN from a set of data is that the number of possible structures is super-exponential in the number of variables. The number of possible combinations  $G$  of DAGs of  $n$  variables can be calculated by the recursive formula [19]

$$G(n) = \sum_{k=1}^n (-1)^{k+1} \binom{n}{k} 2^{k(n-k)} G(n-k). \quad (27)$$

The result of this calculation is given in Table 4 for 1 to 10 variables.

**Table 4: Number of Possible Structures in a DAG**

Variables	Structures
1	1
2	3
3	25
4	543
5	29,281
6	3,781,503
7	1,138,779,265
8	783,702,329,343
9	1,213,442,454,842,881
10	4,175,098,976,430,598,143

<sup>11</sup> Meaning “the structure that most probably created this data” or, more explicitly, “the one representing the joint probability distribution that most closely encodes the dependencies and probability parameters matching those in the data.”

Since the data sets we will be using may have 100 variables or more—and even after pruning have tens of variables—an exhaustive search of all DAGs is clearly not feasible. Therefore, methods that are able to quickly remove large sections of the search space, called approximate searches, are needed.

Many methods of approximate search for best-fit structures have been developed, especially in the last decade [17]. A typical method is to start with one variable, and then add the next best variable (or arc) based on the correlations found in the data set.<sup>12</sup> A common method of measuring the success of these methods is to start with a relatively complicated DAG, use the equivalent JPD to generate a large (10,000 cases) data set, then attempt to re-create the DAG from the data, measuring the number of false or missing arcs. We have used a related method, generating data based on characteristics of a real data set, and then attempting to verify the structure that we believe created those characteristics.

Particularly in Chapter 6: Bayesian Network Algorithm, we will use the characteristics of our problem to efficiently build a BN structure representing the dependencies between the class variable (disease state) and the feature set (ion abundance at each peak position). The primary attribute that will streamline this approach is the unique status of the class variable in a classifier. More general structure learning methods seek dependencies between all variables without specific regard to order or importance; with a classifier, we have a natural starting

---

<sup>12</sup> Some take the opposite approach, starting with all possible arcs and pruning. Another fruitful approach has been to do both, first adding all reasonable arcs and then using further tests to prune unnecessary ones.

point. Even more important, we actively seek to discard all variables that are not directly connected by an arc to the class variable, since instantiation (in our case, measurement) of the set of variables directly connected to the class will cause the remainder to be independent of the class, and therefore, not useful in classification [17].

A *Markov blanket* around a variable is the minimum subset of all other variables that, if instantiated, break any dependence with the remaining variables [17]. Mathematically, for a class variable  $C$  and variable set  $V$ , the Markov blanket around  $C$  is the minimum subset  $S$  of  $V$  which, for all  $X \in \{V \setminus S\}$  (the variables not in  $S$ ),  $MI(X;C | S)=0$ .

In the case of a classifier, we need only find the Markov blanket, if we are sure to measure all the variables in it. Since, in our experiment, each mass spectrum produces specific measurements for each variable, the Markov blanket around the class variable will suffice. It is this specific set of variables we will search for in our structure learning.

### **Parameter Learning**

Once a structure is known, we must determine, empirically or by other means, the values in the probability tables that make up the terms in the JPD. We can refer to them as tables (in the discrete case) because they hold specific probability values for each combination of variables in that term. Thus, for a binary class  $C$  of values  $\{0,1\}$  representing “disease” or “normal,” and a data variable  $A$  with 4

possible values  $\{0,1,2,3\}$ , the network  $C \rightarrow A$  requires two tables:  $P(C=0,1)$  with two entries, and  $P(A=0,1,2,3 | C=0,1)$  with eight entries. These entries are the parameters we seek to determine under the rubric “parameter learning.”

The simplest method (and the one we will use) involves using the maximum likelihood for each parameter given the set of learning data. For a discrete system like the one in the previous paragraph, the maximum likelihood for each entry in the table  $P(A=a_i | C=c_j)$  is just the fraction of training data cases that fall into the bin represented by  $a_i$ , partitioned by class [20]. Thus, if there are 25 total cases in the disease class, and 5 of them fall in bin 1,  $P(A=1 | C=\text{“disease”})$  is estimated to be the maximum likelihood value of 0.20.

Another possible method of calculating the entries would be to use the data to model the underlying population, and then to use the model (a Gaussian for example) to estimate the probability, perhaps by integrating over all the values in the bin. The difficulty with this approach, and the reason that it was rejected for the problem at hand, is that there is no well established model for the expression of proteins in normal and disease groups for the data we wish to examine. Our group was able [21] to determine that the errors caused by sample preparation and instrumentation is fitted well by a log normal distribution. If all cases, for example, had a single true value, the probability distribution could be modeled by the log-normal<sup>13</sup> “instrument function,” since the measured distribution would be

---

<sup>13</sup> Log-normal means that the distribution is Gaussian after taking the logarithm of the raw values..

the single value (a Dirac delta function) convoluted with the instrument function, returning just the instrument function.

We estimated the parameters of the instrument function by studying the QC data described on page 18. At its best, this data has a coefficient of variation of approximately 20%, even after adjustments for laser performance and other experimental factors [21].

Unfortunately, we have no basis for concluding that all normal patients have the same value, or any other specific distribution of values, for the protein abundances we measure. We do observe certain variables to have distributions very close to the log-normal instrument function, with widths approaching the typical variations found in the QC data.

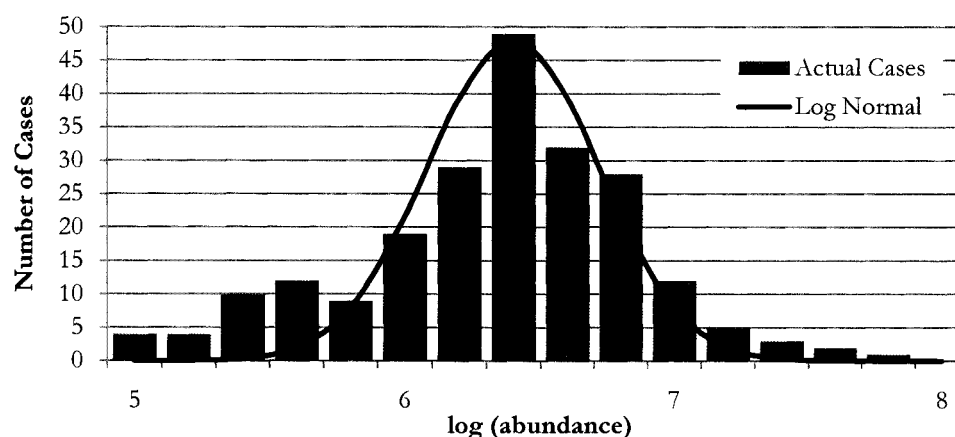


Figure 12: Sample population and instrument function

We conclude this to mean that the underlying population has a distribution of actual abundances that is very sharply peaked, and the convolution of that relatively sharp underlying distribution with the wider instrument function

produces the results we observe. An example from actual data is shown in Figure 12. An approximate log-normal instrument variability function has been added and scaled in height to guide the reader's eye.

## Mutual Information with Class

The scoring criterion we will use to select a feature  $V$  for inclusion in the BN is “mutual information with the class” or  $MI(C;V)$ . This measurement answers the question “how much does knowledge of the value of a specific variable (an ion abundance in our case) tell us about whether or not the person has a disease?”

Figure 13 demonstrates an important problem. Even a completely (pseudo)random data set will include some variables that exhibit mutual information with a random class variable, if enough variables are created compared to the number of samples. The black line in Figure 13 represents the cumulative distribution of MI between each of 200 random variables and a random binary variable designated as the class. A point  $(x,y)$  on the line means that a fraction  $y$  of the 200 variables had  $MI(C;V)$  less than  $x$ . In this experiment about 3% of the variables had  $MI > 0.04$ . One of our similar sized real data sets—the blue line—had about 32 variables with this same MI or higher. We must therefore keep in mind that, with a low enough MI threshold, we are likely to include features that are not truly indicative of a disease, but, due to small sample effects, have matched the class enough times to have been declared as important.



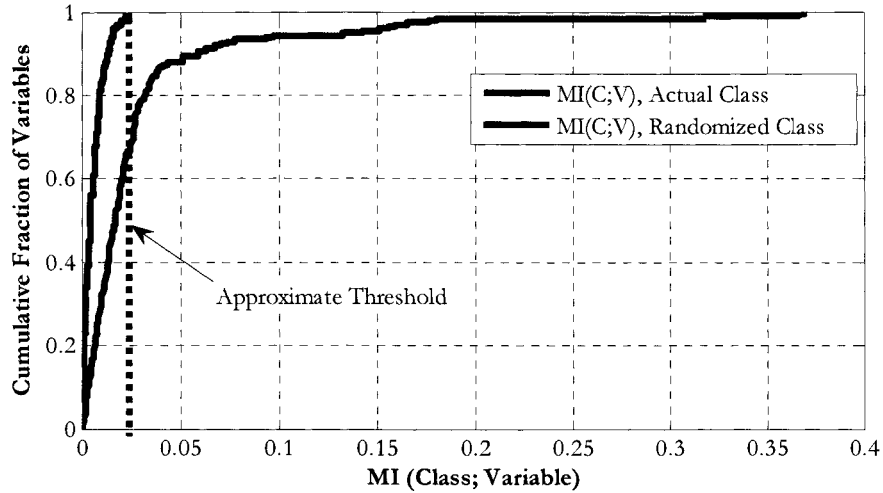


Figure 13: Mutual information threshold

As indicated by the dotted line, we use a value of  $MI(C;V)$  close to the *maximum* value observed between many variables and a randomly generated class as the baseline *minimum* threshold for determining significance. We may need to vary the baseline MI threshold to leave a reasonable number of variables in the selected feature set, while reducing the chance of selecting a non-significant feature. It is still possible with this choice that an important feature has fallen below the threshold and therefore eliminated from further evaluation, or that a random feature has been included.

## Discretization

A Bayesian network can include nodes representing certain *continuous* probability distributions, such as Gaussians, but we have no concrete justification for modeling the features we will examine with one of these allowed distributions. In addition, there are certain limitations to node placement and ordering that continuous distributions impose [17] which we choose not to accept for this problem.

Discrete variables, those that take on a (typically) few specific values, do not impose these limitations in the Bayesian network. In the general case, the possible values that the variable can take may come from sets like {old, middle-aged, young} or {greater than 50.3, less than 50.3}, but we may map these choices to an equal number of integers, e.g. {1, 2} for ease of use.

With the goal of creating a Bayesian Network in mind, therefore, we will discretize, or bin, the values of the abundance variables from a nearly continuous set of values into a few bins. This will cause the values of the variable for a specific case to collapse from a measured abundance like “1283 counts” to “bin #2.” We will later discuss various binning methods and the criteria for choosing bin boundaries, but MI with the class will be an important optimization criterion.

## **Cross-Validation**

Because of the difficulty of obtaining new samples at each step of testing a new classifier, a method of using a single set of previously classified data to both train and score a classifier is used. This is called *cross-validation*.

One such method is simply to train a classifier on the entire data set, and use the resulting model to re-examine each case—as if the result is not known—and score the classifier on its ability to correctly predict the class. We will call the result of such an analysis the *nominal error rate*. The major drawback of this method is that the same data is used to train and test.

As an example of the problem this can cause, consider a researcher sitting at a street corner, recording the type and color of four passing vehicles. For some reason, on that day, two cars that pass by that are red, and two trucks that are blue. He builds a classifier using that data which says “if a vehicle is red, it must be a car.” Instead of testing the model against the next passing car, he tests it against the data he has already collected, finding (incorrectly) that his model is error-free.

The primary way to guard against this problem is to train the model with some subset of the data, then to test it against the remainder. The “leave one out” cross-validation method does just that, training the model under all but one case, then classifying that case with the resulting parameters. By repeating this for all the cases, an overall error rate can be scored. This method is useful because, with a large enough data set, the model parameters are relatively stable as the various “all but one” training sets are selected, and the error rate directly reflects the number of cases classified on this stable parameter space. One problem with the leave one out method, however, is that the stability of the model parameters may be misleading [22].

As an example, consider a classifier that tries to guess the sex of a student based on that student’s height. Unfortunately, the 50-case training set (50% female) happened to include ten members of the women’s basketball team. Using any 49 of the 50 cases will train the classifier that, if the test case is under 5’ 10”, the student is most likely female. The 10 basketball players, all over six feet, will consistently be classified incorrectly. The classifier has an error rate of at least

20%. There is, however, no way to determine whether 20% is a stable measure, as each complete trial of leave-one-out cross-validation produces *exactly the same result*. In fact, it is possible that if 50 new cases were tested, the classifier would have a much lower error rate.

To correct the problem of unknown stability in the error rate, while accepting some increase in the instability of the model parameters, the method of *n-fold* cross-validation is used. In this method, the data set is divided into  $n$  groups, for some small integer  $n$ . One of the groups is held back as a testing set, and the remainder are used to learn the model parameters. The test group is classified and the number of errors recorded. A different group is selected as the test group, and the original test group is returned to the training set. This is repeated until all  $n$  groups have been a test group, and hence, all cases have been classified. In this study, the parameter  $n$  is typically set to 10, and repeated randomized trials made, which decreases variance and increases stability [23]. Table 5 shows an example 5-fold cross-validation, in which each group consists of one-fifth of the total cases.

**Table 5: Example 5-fold cross-validation**

	<b>Group 1</b>	<b>Group 2</b>	<b>Group 3</b>	<b>Group 4</b>	<b>Group 5</b>
<b>Trial 1</b>	Train	Train	Train	Train	<b>Test</b>
<b>Trial 2</b>	Train	Train	Train	<b>Test</b>	Train
<b>Trial 3</b>	Train	Train	<b>Test</b>	Train	Train
<b>Trial 4</b>	Train	<b>Test</b>	Train	Train	Train
<b>Trial 5</b>	<b>Test</b>	Train	Train	Train	Train

To calculate a *cross-validated classification error rate* under  $n$ -fold cross-validation, it is necessary to record the results for each test group after its classification. After all  $n$  trials are accomplished, every case in the data set has been classified exactly once.

At that point a cross-validated error rate can be calculated, using either the probabilistic or deterministic method of classification (see more details on these methods on page 34). We expect that the model parameters would vary more with this method than the “leave one out” method. If, for example, there are 100 cases, with  $n=5$ , 80 of 100 cases are used to learn the parameters, as opposed to 99 of 100 in the “leave one out” method.

This leads to the strength of this method, however—the parameters can be tracked and the variations recorded to better understand the stability of the classifier. Even more importantly, the data can be randomized and a different set of  $n$  groups selected. Since each case is now tested on parameters derived from a different set of training data (unlike the leave-one-out method) it may or may not receive the same classification and the error rate will change after each randomization. By examining fluctuations in the error rate or the feature set selection, we can measure the stability of the classifier, and better predict whether it will continue to achieve similar error rates as new cases are tested.

Repeating the *n-fold* cross-validation in this manner also allows the average error rate of a particular classifier model to be used as a parameter. For example, one might add, to an existing feature set, the feature that decreases the error rate the most. Cross-validation trials in this study are *stratified*, meaning that each training group, while selected randomly, has approximately the same fraction of each class as the overall population. This has been shown to reduce bias and variance [23].

## CHAPTER 5: APPLICATION OF THE NAÏVE BAYESIAN CLASSIFIER

As discussed in Chapter 2, the assumption of independence between features allows a great simplification in the structure and parameter learning of the resulting naïve Bayesian classifier (NBC). A classifier is created by simply building a network with the class variable as the parent node, and the set of features as direct descendants of the class, with no arcs between the features. We have stated that an NBC is a special case of the Bayesian network, and Figure 14 shows this special structure.

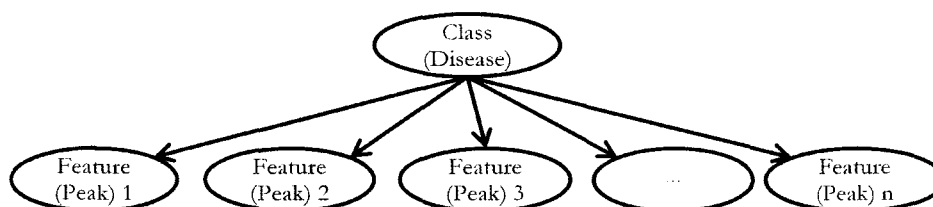


Figure 14: Naïve Bayesian classifier

This type of classifier was used to examine the Leukemia and Prostate Cancer data sets. The first experiment using the NBC was as a simple classifier using all available features, and for this task, it performed well. Performance for feature selection was less successful. Because of the problems associated with the independence assumption, its feature sets were unstable when used as a wrapper for feature selection. While it did select a reasonable feature set, it ignored highly correlated features that were important to further analysis for biomarker discovery. Specific details are found in Chapter 7: Results and Analysis.

## Classification performance

The simplest algorithm one can use to build and test a naïve Bayesian classifier consists of only a few straightforward steps. The first is to discretize the data into two or more bins. Next, for each class, and using the full data set (no cross-validation), find the fraction of samples in each bin, and use this result for  $P(\text{data in bin} | \text{class})$ .

For example, assume there are 100 patients with a class of “disease” and a similar number with a class of “normal,” and a single continuous variable  $X$ . A bin boundary  $x_0$  in the range of  $X$  is chosen by some method, such as fixing the boundary at the entire population’s mean value. Then a second variable  $X'$  (a discretization of  $X$ ) is created, with  $X'=0$  if  $X < x_0$  and  $X'=1$  if  $X > x_0$ . The parameter  $P(X'=0 | \text{Class}=\text{disease})$  is estimated by counting the number of samples labeled “disease” in the lower bin and dividing by the total number in the class (100 in this example). If 80 samples of the disease class have  $X > x_0$ , then  $P(X'=1 | \text{disease})$  is estimated to be 0.80.

After these two steps, Bayes’ Theorem (10) is invoked, using the estimated parameters, to find the result for  $P(\text{class} | \text{data values})$ . A typical result might read “the probability that this person has the disease, given that the ion abundance measured was in bin  $x$ , is 75%.”

A classification threshold is used to convert this probability into a specific class, such as the value 0.40 in “if  $P(\text{disease} | \text{Data}) > 0.40$ , the patient is classified as

having the disease.” The threshold can be adjusted depending on the relative importance of eliminating *false positives* or *false negatives* (see Glossary). This results in a deterministic classifier (see page 34). For the work presented here, the threshold is always set to 0.5.

In the simple one variable example described above, it is likely that each of the 80 samples from the disease class with  $X > x_0$ , will be classified correctly as “disease,” while the other 20% will be classified in error. If the normal class happens to have a similar result (80 correct, 20 in error) we say the classifier has a 20% nominal error rate.

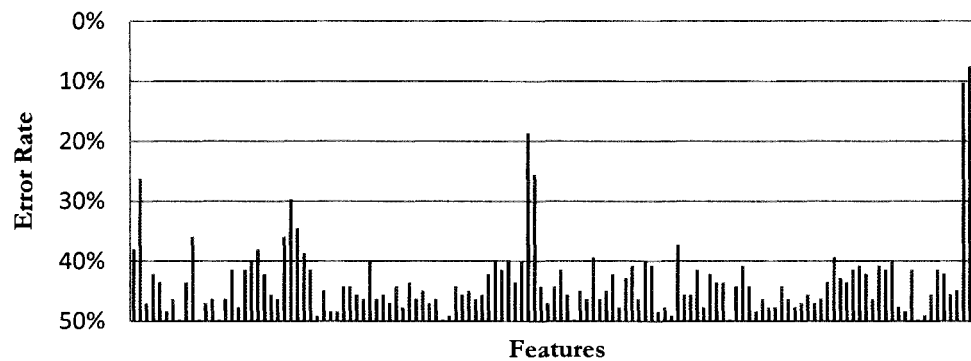
This error rate changes under cross-validation (see page 57 for details on cross-validation methods). Under a 10-fold cross-validation, for example, 180 of the 200 samples will be chosen as a training set, the parameters such as  $P(X=0 | class=disease)$  learned from that subset, and the results used to find  $P(class=disease | data)$  for the remaining 20 samples. The test group is returned to the training group, a different group of 10% of the samples is used as a test group, and the process is repeated (with new parameters calculated) until all the samples have been classified once. A probability threshold for declaring the class is chosen, and an error rate calculated.

In this experiment, the values  $X$  are the measured abundances at each peak position. The example above is extended to many variables using the independence assumption, as discussed in the section leading up to equation (13).



Many trials of  $n$ -fold (with  $n$  typically set to 10) cross-validation are run, with the samples included in each of the  $n$  groups randomized between trials. This allows us to find an “average” cross-validated error rate, as well as determine the variability of that error rate between trials.

Figure 15 shows the results of nominal error rates using each feature—by itself—to classify the cases in the 199 peak leukemia data set. A classification error rate (also called a misclassification rate) of 50% means that a feature was no better than a random choice at choosing the correct class. An error rate of 0% means that feature was a perfect predictor of the class. Any of the three peaks at the far right of the figure, representing mass-to-charge ratios near 11700  $m/z$ , predict more than 90% of the population correctly.



**Figure 15: Nominal classification error rates of individual variables**

From a pure classification standpoint, the minimum single-variable error rate (about 10% in this data) provides a metric by which to compare any multi-variable feature sets. We expect, however, that classification under a cross-validation

scheme, with the possible exception of the “leave one out” method [22], to give a somewhat higher—and variable—error rate [23].

## Feature Selection

The reader may note in Figure 15 the several clusters of “diagnostic” peaks—those that by themselves may indicate the classification of the sample. The peaks are shown in increasing  $m/z$  value, so that variables that are adjacent in Figure 15 are often only a few tens of Daltons apart in molecular weight. These are often a primary peak and its adducts or modifications; if a peak is diagnostic, its adducts appear to also be diagnostic. On the far right of the chart, for example, is a series of three of the most diagnostic peaks, which represent the  $m/z$  values of 11684, 11727, and 11740  $m/z$ .

Because of the large number of variables, exhaustive search methods of various combinations of features are impractical. Instead, features are selected via forward selection and backward elimination (See Chapter 4: Classification and Feature Selection). The MATLAB<sup>14</sup> code that accomplishes this is found in Appendix B: MATLAB Code. This code contains a wrapper, a method of feature selection where the features are selected by their utility under cross-validation.

---

<sup>14</sup> MATLAB® (for “Matrix Laboratory”) is a commercial program by The Mathworks, Inc. Version 7.7 was used for this research.

## Error Rates

Figure 16 below shows the effect of forward selection on error rate. To select the next feature  $k+1$ , all features not previously selected are added one at a time to the  $k$  features that have been permanently selected to that point. The new group of features that gives the lowest average error rate over a number of repetitions then becomes the new permanent feature set. The data come from the Prostate Cancer data set (described in Chapter 3), and show four typical trials of selecting 20 features.

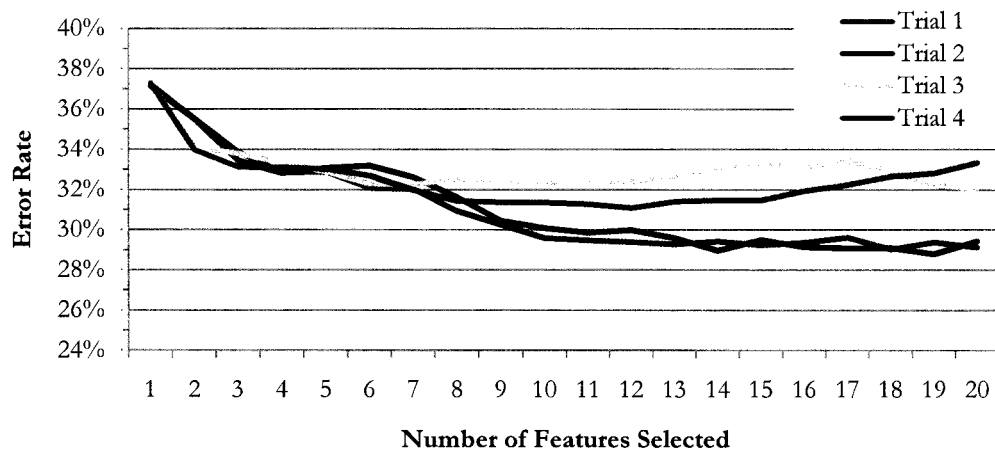
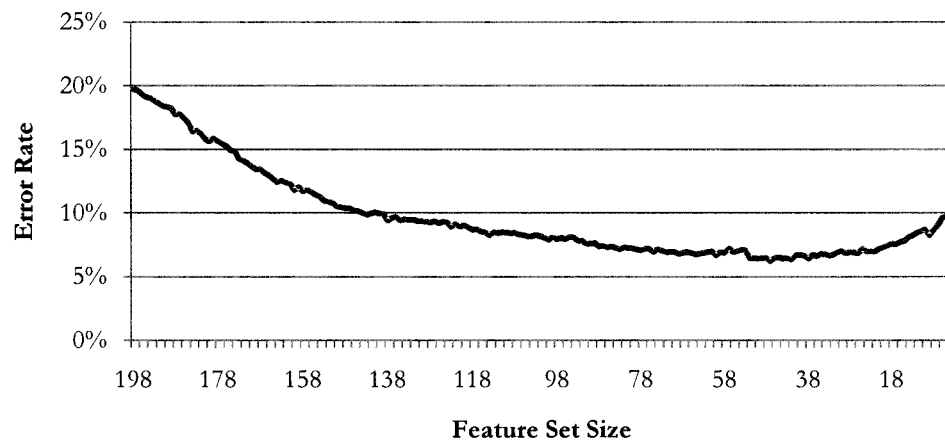


Figure 16: Cross-validated error rate, forward selection, PCA data

The error rate rises in the middle of Trial 3, even though the search is for the lowest error rate. This is because the selection of another variable is *mandatory*, insomuch as the algorithm was set to select exactly 20 variables. It happened in this trial that, after selecting variable 7, the next best variable's addition increased the error—but it increased it less than any other selection.

A trial with backward elimination shows a similar trend. In Figure 17 below, all but 5 of the variables in the 199 peak Leukemia data set are removed. The error rate is reduced from 20% to 6.5% after removal of the first 150 features, and then rises from there as additional features are removed. Detailed results from each data set are found in Chapter 7: Results and Analysis.



**Figure 17: Error rate during backward elimination, Leukemia data**

This rise in error rate allows one to estimate the size of an optimum feature set. The error rate found in the trial above reaches a minimum at about 50 features. This minimum is consistent; over several trials, the cross-validated error rate dropped to between 6% and 6.5% at about 50 features.<sup>15</sup> If the search were intended to find this number, these results would be satisfactory. However, we seek to find a specific *set* of features, and therefore must investigate which features are producing this error rate. If the same 50 features are selected under large numbers of independent cross-validations, the feature set is said to be stable, and

---

<sup>15</sup> The nominal error rate with 50 features was approximately 4.8%.

that feature set would be expected to perform similarly on the next sample of unknown classification.

However, if we find that the feature set is unstable under cross-validation, then we must investigate further.<sup>16</sup> This requirement turns out to be the fatal flaw in the NBC for the purpose for which we intended it.

### Feature Set Stability

To illustrate, consider the set of “the first 50 features selected” during forward selection of the Leukemia data. If we repeat this trial a number of times, we can determine the stability of the technique by examining the percentage of times a specific feature is selected. A perfectly stable system would select the same 50 variables  $m$  times in  $m$  trials, however, that is not the result obtained.

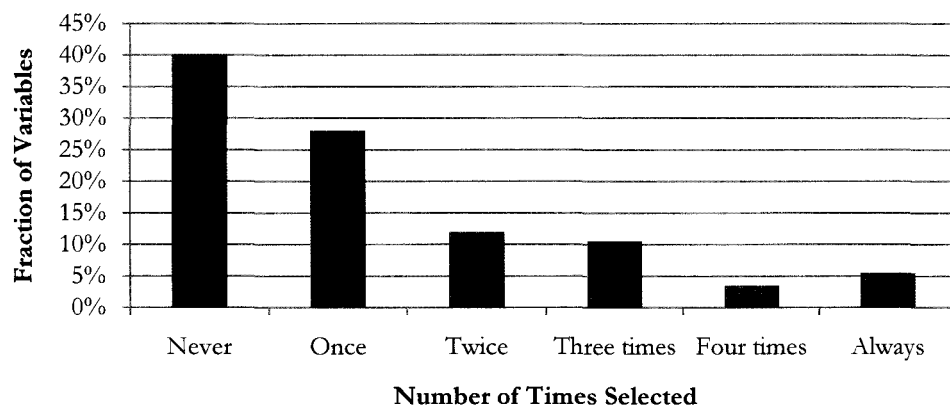


Figure 18: Variable selection frequency, 5 forward selection trials

---

<sup>16</sup> The search for this instability is a key reason for using  $n$ -fold cross-validation over “leave one out.” Since the training data is nearly constant under “leave one out,” we would expect this instability to be masked, at least until a set of additional cases are attempted to be classified.  $n$ -fold allows the researcher to predict how well the training data will model another sample set. See page 31.

Figure 18 shows the result of selecting the best one-fourth of the variables in each of five trials. While the goal is that the same 25% of diagnostic variables would be selected during each trial, but in fact 60% of the variables are selected. Five percent of the (diagnostic) features are selected consistently; the majority of features, however, are selected only occasionally.

Table 6 lists the peaks selected 4 or 5 times during this trial. We will compare such lists across different experiments in Chapter 7.

**Table 6: Features selected frequently during forward selection**

<b>5 Times</b>	<b>m/z</b>	<b>4 Times</b>	<b>m/z</b>
3	2798	38	3768.5
49	4031	57	4211.0
84	4982	64	4372.1
113	5997	103	5751.9
124	6562	104	5773.8
144	7649	122	6515.9
151	7863	132	6851.0
152	7888		
193	10547		
198	11697		
199	11742		

*Leukemia Data, 199 Variable Data Set*

Part of this variation comes from the process of cross-validation, since at any one step, if two variables are approximately equally diagnostic, the randomized selection of training and testing groups will affect which is actually chosen for inclusion or elimination. We have attempted to mitigate this variation by repeating the cross-validation a number of times (we usually choose 30) and using the mean error rate produced as a selection metric.

A secondary source of variability is that we may attempt to select variables after all diagnostic features have already been chosen. In this case, we would expect diagnostic features to be chosen consistently, and non-diagnostic features to be chosen occasionally. We therefore note those features chosen consistently for further consideration.

### Effect of Correlations

Despite the averaging of large numbers of cross-validated trials, we noticed large variations in feature set selection that were clearly problematic. For example, in the 4 PCA data forward selection trials whose error rates were presented in Figure 16, the specific feature selections vary significantly. Table 7 shows the first 10 features selected during those trials.

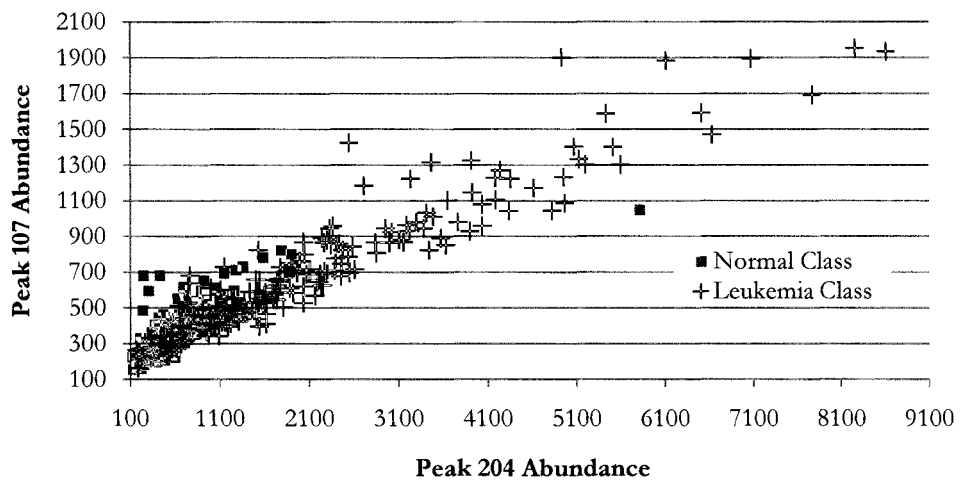
**Table 7: Features selected on separate trials (PCA data)**

<b>Trial 1</b>	<b>Trial 2</b>	<b>Trial 3</b>	<b>Trial 4</b>
23	23	43	43
6	6	66	66
44	44	41	41
90	90	85	63
84	7	73	15
2	1	82	14
1	45	11	81
66	66	37	1
45	13	15	77
31	85	10	93

During the selection of the first seven variables, the error rate curves in Figure 16 are relatively consistent, although the actual features selected are almost completely different. The first several selections in trials 1 and 3 yield significant drops in the error rate, but do so with different variables. Furthermore, those variables selected

first in one trial are rarely selected later in another. We noted that this phenomenon occurred often among variables with high linear correlation.

As mentioned previously, such features exist in our data. The  $m/z$  values of peaks 204 and 107 in the final Leukemia data set have a ratio of almost exactly two, indicating the possibility that peak 107 is a doubly charged ion ( $z=2$ ) of peak 204. In fact, the linear correlation between those two features is 0.952. A scatter plot of the two features, with a marker for each of 417 replicate spectra, is shown in Figure 19.



**Figure 19: Correlation between two diagnostic peaks**

To investigate further, we created a simple generated data set that was intended to determine the effect of highly correlated features on the selection process.



## Correlation Effects

To replicate the problem of correlated features, this generated data set was given three primary diagnostic<sup>17</sup> features. Those three features were replicated elsewhere in the data in two ways—first by simply inserting a duplicate (perfect correlation) of each diagnostic feature, and second by using the values in the original, plus Gaussian noise, to create a more weakly diagnostic (but correlated) feature.

Figure 20 shows a heat map of the artificial data with the three perfectly correlated pairs. The most diagnostic features are labeled 10, 20, and 30, and their duplicate features are labeled 5, 15, and 25, respectively. In the heat map below, the first 100 cases (rows) represent one class, the remainder, the second class. The diagnostic features are clearly separated between these two groups.

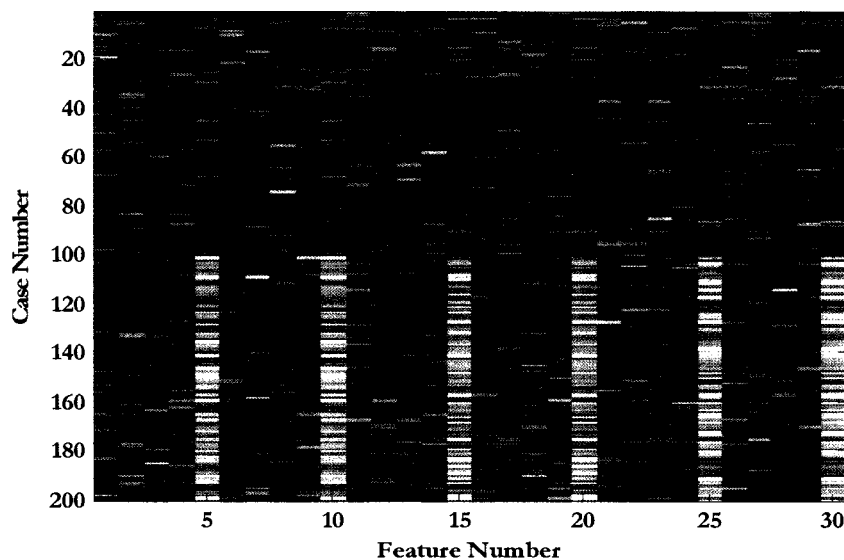


Figure 20: Generated data set with perfectly correlated features

<sup>17</sup> To make them “diagnostic,” each feature was generated from a normal distribution with a mean value in one class that was separated by one standard deviation from the mean in the second class.

Forward selection of 10 features is attempted on both the perfect, and the noisy, data sets. Error rates for both data sets are low. However, after repeated trials the effect of highly correlated features becomes clear: after one feature is selected, the second feature becomes much less important of an addition to the feature set. For the data set with the correlated, but noisy, features<sup>18</sup> (Figure 21), the error rate drops rapidly when an independent diagnostic peak is selected, but less so when additional, but correlated, peaks are included.

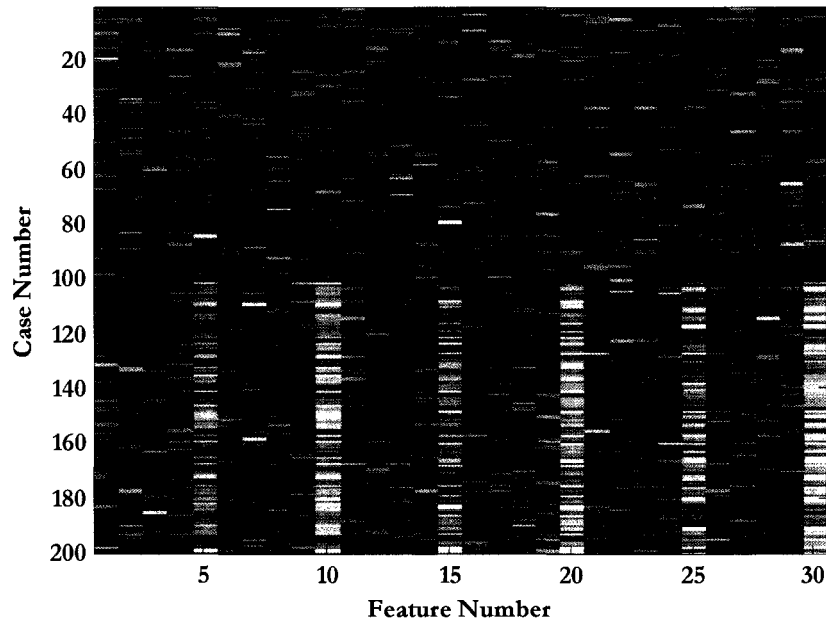


Figure 21: Generated data set with correlated features with noise

Table 8 shows the first 10 features selected for this artificial data with noise, and a typical error rate profile. The algorithm selects one feature from each correlated pair (5 and 10, 15 and 20, or 25 and 30), and then a second feature is selected from 2 of the 3 pairs, but not from the third. In fact, feature 5 is never selected, even

<sup>18</sup> While the majority of the linear correlations among the random variables were less than 0.1, between the 3 diagnostic pairs the linear correlations were 0.72, 0.69, and 0.75.

though other, purely random, variables are selected. Correlated features are highlighted in red.

Table 8: First ten features selected, generated data with correlation

Order Selected	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Error Rate
<b>1</b>	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	11.4%
<b>2</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	8.2%
<b>3</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	5.3%
<b>4</b>	<b>20</b>	<b>20</b>	<b>20</b>	<b>20</b>	<b>20</b>	4.1%
<b>5</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	3.3%
<b>6</b>	1	2	2	9	29	3.2%
<b>7</b>	7	12	16	12	2	3.1%
<b>8</b>	8	18	8	13	18	3.1%
<b>9</b>	4	3	18	18	12	3.1%
<b>10</b>	2	27	12	19	11	3.1%

Upon repeating the experiment with the other set of generated data, in which the correlated pairs (e.g. 5 and 10) are exact duplicates of each other, the problem becomes even more apparent. One of the two features is selected, apparently at random, from each pair, but the other is never selected.

Table 9: First ten features selected, generated data with duplicates

Order Selected	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Error Rate
<b>1</b>	<b>30</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	11.1%
<b>2</b>	<b>10</b>	<b>5</b>	<b>10</b>	<b>10</b>	<b>5</b>	8.0%
<b>3</b>	<b>15</b>	<b>15</b>	<b>20</b>	<b>15</b>	<b>15</b>	5.8%
<b>4</b>	6	17	7	7	16	5.7%
<b>5</b>	4	8	12	4	6	5.6%
<b>6</b>	8	4	28	23	4	5.6%
<b>7</b>	11	9	17	8	7	5.5%
<b>8</b>	3	11	4	11	13	5.2%
<b>9</b>	9	3	23	9	8	5.0%
<b>10</b>	2	28	26	3	17	4.7%

It is possible to show why this occurs mathematically. Consider a discrete random variable A with alphabet  $\{1,2\}$ , and a second variable B that is a duplicate of A, so

that  $P(A=i|B=j) = \delta_{ij}$  (Dirac delta). They are used to classify a class variable  $C$ , also with alphabet  $\{1,2\}$ .

In the case of classifying  $C$  using only variable  $A$ , Bayes' Theorem yields

$$P(C = 1|A = i) = \frac{P(A = i|C = 1) P(C = 1)}{\sum_k P(A = i|C = k) P(C = k)} . \quad (28)$$

If we classify  $C$  using both  $A$  and its duplicate  $B$ , Bayes' Theorem gives

$$\begin{aligned} P(C = 1|A = i, B = j) &= \frac{P(A = i, B = j|C = 1)P(C = 1)}{\sum_k P(A = i, B = j|C = k)P(C = k)} \\ &= \frac{P(A = i|C = 1)P(C = 1) \delta_{ij}}{\sum_k P(A = i, |C = k)P(C = k) \delta_{ij}} , \end{aligned} \quad (29)$$

which (because it never occurs that  $i \neq j$ ) is the same as the one variable classification. Because the classification  $P(C=k|\text{data})$  has the same probability whether  $A$ , or the duplicate pair  $\{A,B\}$ , are used, the error rate cannot decrease if  $B$  is added to the feature set that already includes  $A$ .

Again, this might not be a significant problem if the objective was simply to build a classifier, since selection of either  $A$  or  $B$  is "good enough." However, our goal is to determine not only a minimum feature set, but rather the set of all features that provide significant information about the disease state. More discussion on this can be found in Chapter 6: Bayesian Network Algorithm. A further discussion of the NBC instability problems is found in Appendix A: Mathematics.

However, the NBC is useful in that it provides a check (of feature selection), and bound (of error rates), for further investigation.

## CHAPTER 6: BAYESIAN NETWORK ALGORITHM

Given the feature selection problems associated with the NBC, it was necessary to expand the methodology to consider links between variables. This naturally leads to the creation of a two-level Bayesian network.

### Goal

Because we only seek a Markov blanket of variables for classification (see page 51), that final DAG will look very much like a NBC, but with the possibility that arcs will exist between two variables that are each a child of the class variable. In order to identify variables that are correlated to diagnostic variables (which is what the NBC failed to do) more robustly, we will also seek a second level of variables. These are variables not connected to the class except through another variable. Figure 22 is an example of this type of structure.

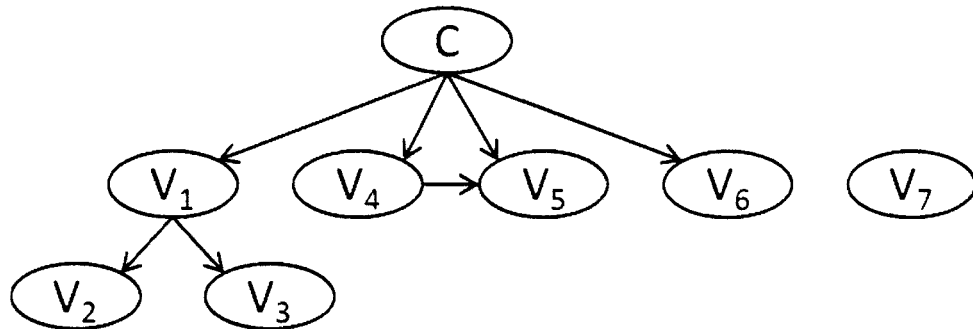


Figure 22: Bayesian network for MS data

In this network,  $C$  represents the disease state, or class variable. The  $V$  nodes represent peak features.  $V_7$  has been determined to be independent of the class

and will be removed from the variable space.  $V_6$  is a first level variable, as are  $V_4$  and  $V_5$ ; the latter two have been determined to be connected to each other. We will refer to this as a *triply connected structure* or *triple connection*.  $V_1$ , a first level variable, is connected to two children,  $V_2$  and  $V_3$ . We seek to build such a network from the MS data sets, noting the first and second level features.

Furthermore, knowing that we will encounter two variables that represent a single physical molecule being measured in separate variables, we will attempt to increase the classification ability by finding and recombining the values of such variables.

### Algorithm

The algorithm that creates the two level BN from the data set is enclosed in two loops: an outer one to repeat the entire process several times to suppress statistical fluctuations, and an inner one for cross-validation. Inside that second loop, where a training group and test group have been selected, the three primary steps occur: structure learning, parameter learning, and classification. Classification of the test group is concatenated until all test groups (and hence the entire data set) have been tested by a network, so that an overall population error rate is available for each full cross-validation.

The full code for the algorithm is presented in the Appendix; the simplified pseudo-code is:

```
pre-process the data;
for each of r repetitions
  build n cross-validation groups;
  for each of n sets of training and test groups
```

```

%      Structure learning section
      find all significant connections;
      test to prune and direct arcs;
      attempt to combine linked variables;
      record resulting network;
%      Parameter Learning section
      find probability tables for first level arcs;
%      Classification section
      classify the test group using the parameters;
      save the test groups predicted classification;
      chose another training-test group sets until complete
      check all predicted class values against known values;
      record error rates;
randomize and repeat.

```

## Initial data processing

Before the start of the algorithm, the data is processed according to several options available to the user. These include replicate averaging, total ion normalization, and the removal of cases with extremely low signal-to-noise ratios.

## Structure Learning

To learn the structure, we exploit the assumption that the class variable has a unique position in the structure, since it has no ancestors, and variables with no path to the class are to be discarded. To determine which variables are connected to the class, we need a scoring criterion. Several scoring criteria have been used in recent research; one popular criterion is the Bayesian Information Criterion (BIC) [17]. The majority of methodologies that use scoring functions like BIC do so to compare structures. Two structures are compared, such as a baseline structure and the same structure with one additional arc added. Such methods often heuristically build a structure from a set of completely unconnected (or completely connected) nodes. Given our goal, we require a much simpler set of tests—those necessary to find connections to the class, and connections between variables. We can then

prune all connections with more than a two-node path to the class, creating a network like that illustrated in Figure 22.

The method we employ is somewhat similar to that of the *Chow-Liu tree* described in Jensen [17], p. 250. Chow and Liu [24] described a method of building a *maximum weight spanning tree* (MWST), wherein each arc is assigned a weight by some method, and a network of maximum likelihood results.<sup>19</sup>

The Chow-Liu tree allows flexibility as to the method of determining the weight of each possible arc between variables. The robust ability of mutual information to determine informational correlations between variables is well suited for this requirement, and has been used previously; a MATLAB library is available [25].

The method of Jensen and others builds a MWST among all variables in the network, only finally choosing a specific variable as the root node. One difficulty with directly applying previous structure learning research to our problem is that we seek to limit the complexity of the resulting network to one that matches our understanding of the underlying biological processes.

We have therefore simplified that method, and reduced the computational expense significantly, by starting with the class and building a modified *Chow-Liu tree* to two levels. The result is also similar to a *classification tree* as discussed briefly in Jensen [17].

---

<sup>19</sup> The likelihood for network  $\mathcal{B}$  on a data set  $D$  is  $P(D|\mathcal{B})$ .



## Mutual Information

Due to the strength of correlations between variables, the MWST method fails to account for the importance of the class variable in our search. We therefore choose to maintain the method of assigning weight via mutual information, but do so starting directly with the class node. This method requires a test to determine “when to stop” attaching arcs from the class to the features.

To provide a threshold for mutual information score that we will consider significant, the algorithm takes the actual data, and for each variable, computes mutual information between that variable and a vector created by randomly permuting the class variable. This process is repeated a number of times and the maximum<sup>20</sup> mutual information found is used as the baseline for the minimum significant mutual information.

While this threshold would seem to provide the “highest MI between a variable and the class expected for a non-diagnostic feature,” in one of our data sets, dozens of features exceeded this threshold. The noise introduced by the various instrumental and chemical preparation processes, particularly that of signal pre-processing, results in a spread of mutual information values that far exceeds that which one would expect.

Figure 23 shows a histogram of mutual information between variables and the actual class for the Leukemia data, as well as the same data with a randomized

---

<sup>20</sup> We take the 99<sup>th</sup> percentile of more than a thousand trials as the “maximum.”

class. These results represent the data discretized by a simple {high, low} method; upon further optimization the number of features exceeding the threshold increases farther.

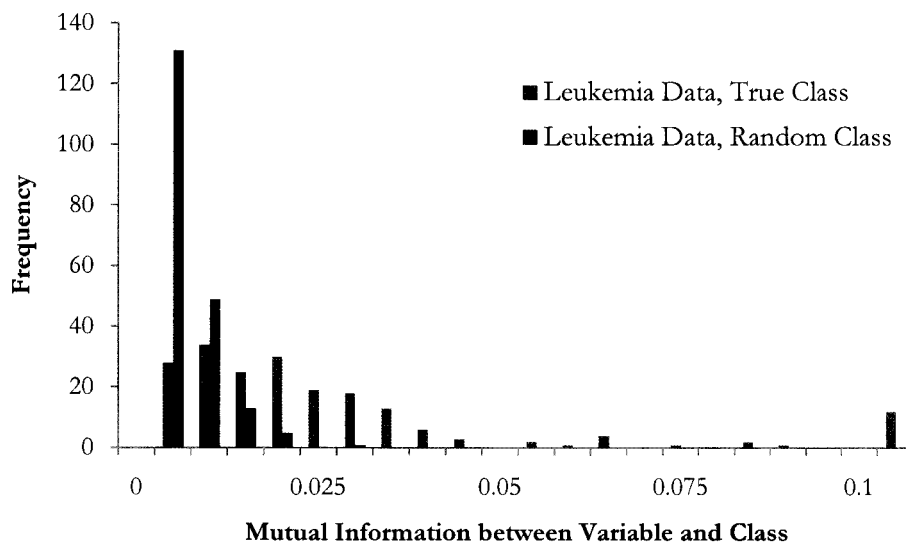
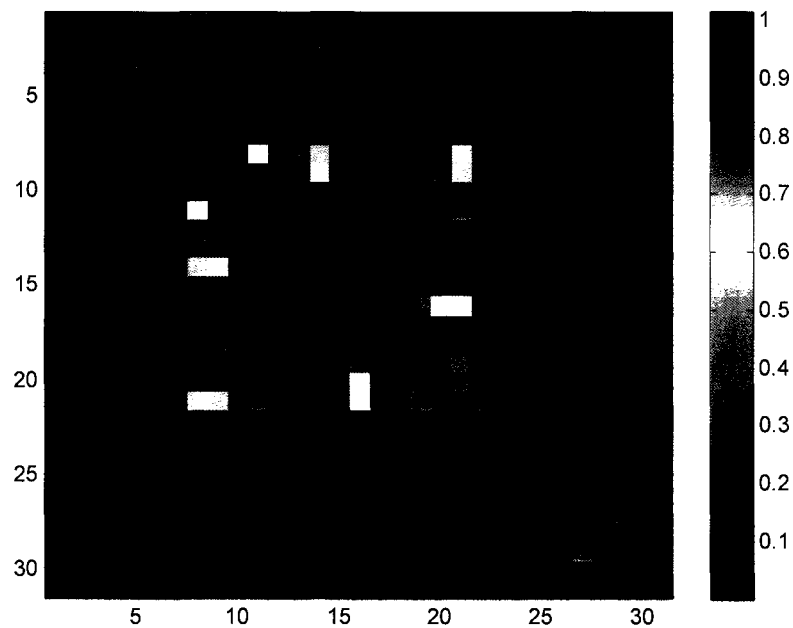


Figure 23: Histogram of MI between features and the class

While we might expect a single molecule to show up as many as ten  $m/z$  positions (including multiply charged states, adducts, and modifications), we do not expect many proteins to be markers for a given disease state [2].

We verified this result using the QC data set. While no “class” exists, since all samples are identical, we can examine mutual information between variables, and set a threshold with a randomly created class assigned to each case. Where a feature is physically related to another, such as an ionization satellite, we would expect to see high mutual information between variables. For the majority of the 435 possible unique combinations of the 30 variables we would expect little

mutual information, since all variations in the data are purely from the sources of noise listed previously. The heat map in Figure 24 shows that large areas of significant mutual information are present.<sup>21</sup> We expect some high MI scores, such as the score of nearly 1.0 between features 8 and 9, which is likely the result of an adduct. The large area of high MI in the center region, however, is more likely to be a signal processing error rather than a physical or biological process. Much of these MI(variable; variable) scores exceed the values of MI(class, variable) in our data sets and would mask the structure we seek under a MWST method.



**Figure 24: Mutual information between variables, QC data**

---

<sup>21</sup> Assignment of a randomly chosen class to each case allowed a threshold of approximately 0.05 to be considered significant.

Given these indications that a randomized MI threshold would underestimate a significance level, the algorithm was modified to apply a multiplicative factor to the MI threshold.

For each data set, we examined the effect of changing this threshold factor on the cross-validated error rate and the number of variables selected to be directly connected to the class variable. In the Leukemia data set, factors below 2.0 (double the “randomized” threshold) resulted in an unrealistic number of first level features selected, as well as an error rate significantly higher than the results of previous analyses. However, when the threshold was set to approximately 3.4, the error rate reached a minimum, and a biologically reasonable number of features were selected.

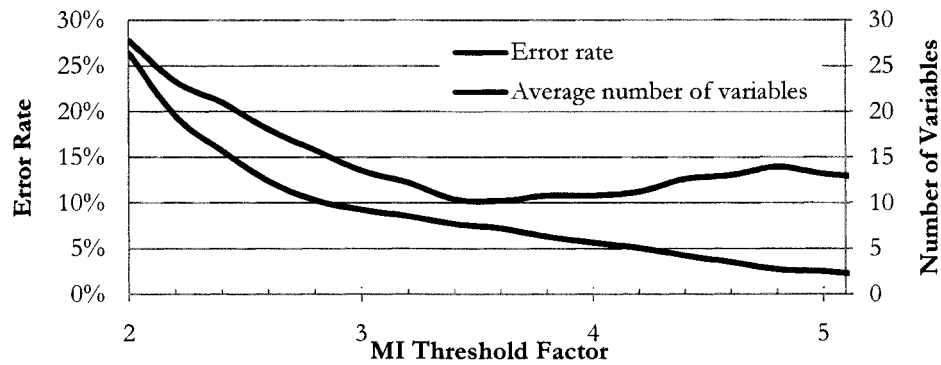


Figure 25: MI threshold effects under 10-fold cross-validation

### Adjacency Matrix

While discovering the BN structure, we need a method to encode the various arcs and nodes efficiently. Our method, following that of the *Bayes Net Toolbox* open

source MATLAB library written by Kevin Murphy<sup>22</sup> (with contributions), is that of the *adjacency matrix*. In general, with  $n$  nodes in a structure, the adjacency matrix is an  $n$ -by- $n$  logical array. The  $(i,j)$  entry in the array represents the truth value of the statement “an arc exists between node  $i$  and node  $j$ .” Care must be taken in its assembly, as the requirements for a DAG are stricter than those of the adjacency matrix—for example, a *true* in a diagonal element represents an arc from a node to itself, which is not allowed in a DAG. An adjacency matrix for Figure 10 is shown below.

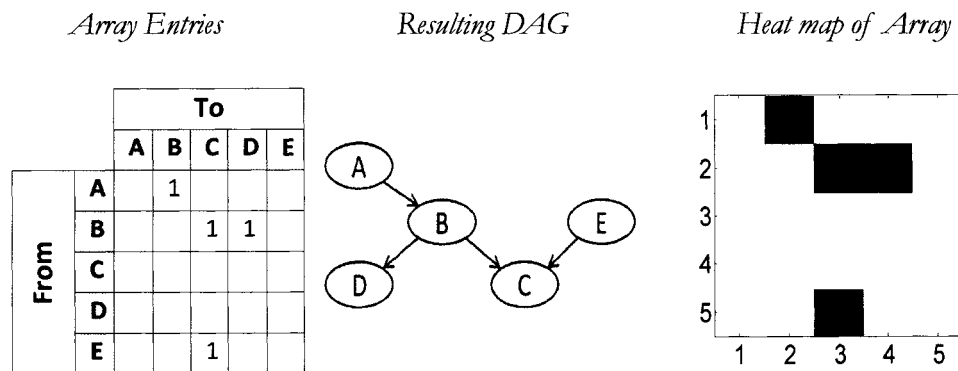


Figure 26: Adjacency matrix representation

### Discretization

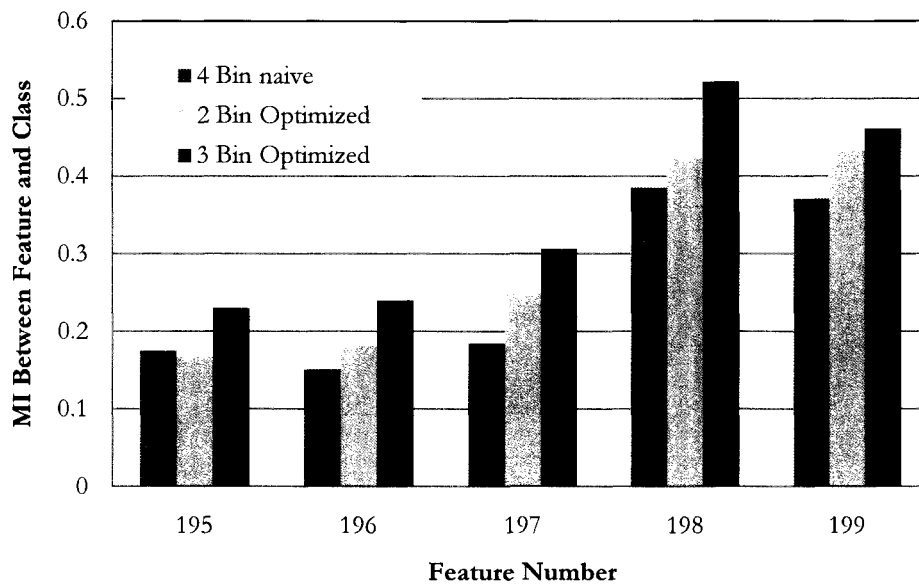
The first step in the structure learning is to find arcs from the class variable to the peak variables, or  $C \rightarrow V$ . For the reasons mentioned previously, the abundance values need to be discretized.

Several methods of discretization of the variables were explored. One method, which we termed as *naïve binning*, models the entire population as a normal

<sup>22</sup> The BN Toolbox is hosted at <http://www.cs.ubc.ca/~murphyk/Software/BNT/bnt.html>.

distribution, and uses combinations of the mean and standard deviation to set bin boundaries. A second method attempts to optimize  $MI(C;V')$ , where  $V'$  is the value of  $V$  discretized over the boundaries. Both two bin, with a single central boundary, and three bin, with two central boundaries  $l$  and  $r$ , were used. The class variable is already discrete (disease or normal).

An example taken from several features of the Leukemia data set is shown in Figure 27. Three bin optimized discretization enhances mutual information compared to naïve methods and two bin optimization.



**Figure 27: Results of optimizing discretization boundaries**

While it is possible that mutual information could be increased further with increasing numbers of optimized bin boundaries, practical limits exist. First, the search for optimal boundaries is computationally expensive. Second, the greater the number of boundaries, the more likely it is that small sample effects, such as central bins having no cases from one class, will occur.

### Three Bin Discretization

Three bin discretization was therefore chosen for this algorithm. If the variable is diagnostic of the class, central values that are common to both classes are isolated in the center bin, and the probability difference between classes falling in the outer bins is maximized.

Figure 28 demonstrates this effect. In that figure, a variable in the generated data set is separated into classes, and each class is histogrammed by the measured abundance value. Bin boundaries (green dashed lines) are placed to isolate the area where class is the most uncertain. In the outer bins, the probability ratio  $P(\text{bin} | \text{class}=1) / P(\text{bin} | \text{class}=2)$  is far from unity, providing the best possible input for a Bayesian analysis.

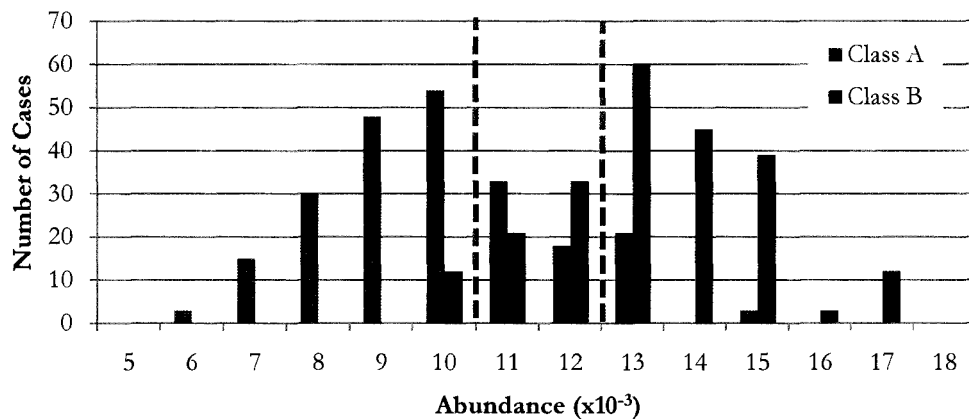
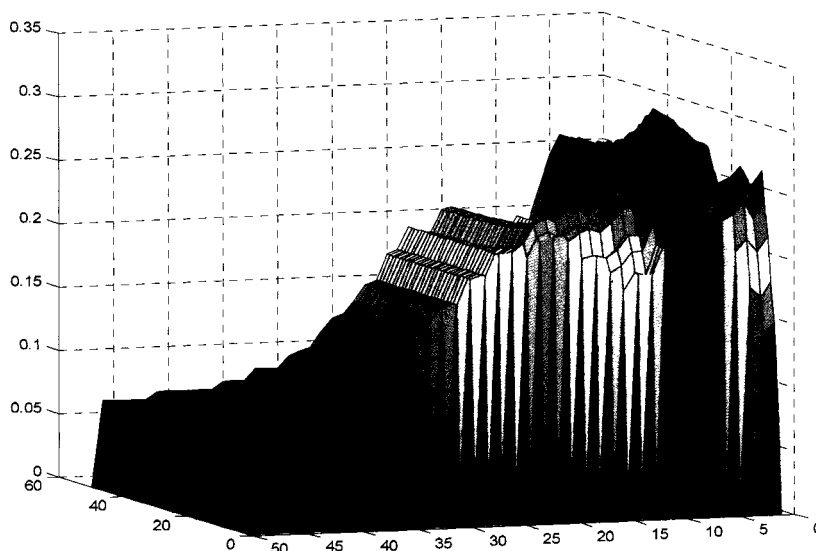


Figure 28: Center bin isolates uncertainty

## Boundary Optimization

The algorithm finds boundary values  $l$  and  $r$  lying between  $\max(V)$  and  $\min(V)$  using a 2-D exhaustive search.<sup>23</sup> Results of one such search (feature 197) are presented in Figure 29. The values on the lower axes represent possible positions of  $l$  and  $r$  as they are stepped across the range of  $V$ . The vertical axis represents the mutual information  $MI(C;V)$  resulting from discretizing on those boundaries.



**Figure 29: Search for optimal boundaries for three bin discretization**

The maximum point represents the  $(l, r, \max MI)$  point which is used for further testing. All cases (training and testing groups) are discretized on the  $l$  and  $r$  boundaries found in this search.

---

<sup>23</sup> The exhaustive search is inefficient but simple to implement. The computational expense was minimal, as optimization is only accomplished once per cross-validation.



### First Level Connections

Once discretization is complete, we score all  $C \rightarrow V$  connections and note those with  $MI(C;V) > \epsilon$ , where  $\epsilon$  represents the mutual information threshold found from random data, adjusted as described on page 82, to reduce false arcs. Connections meeting this test are entered in the appropriate element in the adjacency matrix.

### Second Level Connections

The second level should consist only of connections of the type  $V \rightarrow W$ , where  $V$  is a node found in the first level, and  $W$  is a node (also representing a peak position abundance variable) found using the test  $MI(V;W) > \epsilon'$ .

$\epsilon'$  found in this test is scaled from the  $\epsilon$  used previously, since the maximum value of mutual information differs according to the number of elements in the alphabet of each variable. For example, if the class has two possible values, and the variables three, the maximum of  $MI(C;V)$  is  $\log_2(2)=1$ ; the maximum of  $MI(V; W)$  is  $\log_2(3)$ . For the leukemia data,  $\epsilon$  was about 0.05,  $\epsilon'$  was about 0.08.

Initially, the entire adjacency matrix is filled with the results of this test. However, the algorithm then clears irrelevant arcs, such as those between variables with no path to the class and those more than two levels beneath the class. This removes triply-connected structures on the second level. Triply-connected structures may remain on the first level, with an unknown arc direction between the features, since  $MI(V;W)$  is symmetric.

### Parent-Child Identification

We must address the triply-connected features to determine if they are a result of a true triple connection (which is not allowed in a Chow-Liu tree) or if one of the nodes is a child of the other—but not the class. We seek to direct or remove arcs in the manner shown in Figure 30.

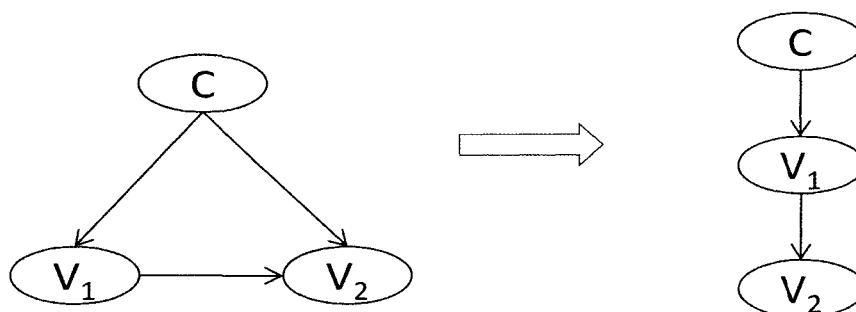


Figure 30: Removal of false connection to class

We believe we will find such structures based on our understanding of the physical processes involved. In the diagram above,  $V_1$  might represent a primary ion and  $V_2$  a modification of that ion. Strictly adhering to the causality precepts for the BN, we might instead seek to insert a hidden variable between the class node and the inverted V arcs in the left side of Figure 30, with the hidden variable representing the base molecule before ionization. However, the result will be the same, as will be discussed in the next section on metavariable creation.

To test the triply-connected structure with the goal of simplifying it into a serial connection, we use a property of the Bayesian network. In the serial structure, if the center variable is known, the root variable and the child variable become independent (see the discussion on page 45). Mathematically,  $MI(C;V_2|V_1)=0$ .

We will not require the mutual information to vanish, due to the inherent noise in our data—rather we test for a significant drop such that  $MI(C;V_2|V_1) \ll MI(C;V_2)$ .

Specifically, we test that

$$\frac{MI(C;V_2) - MI(C;V_2|V_1)}{MI(C;V_2)} \geq \delta, \quad (30)$$

where  $\delta$  represents some threshold. It is important to note that, for testing as parent and child, the arc between the two variables must already be directed since  $MI(C;V_2|V_1)$  is not symmetric in  $V$ . We do so by choosing the greater of the terms  $MI(C;V_i|V_j)$ , where  $i$  and  $j$  are the indices of the two variables to be tested and are permuted. Thus, whichever variable maintains a stronger correlation to the class upon instantiation of the other is chosen as the parent.

The reader who is familiar with the controversy regarding the “monotone DAG faithfulness” assumption, made by Cheng, et al.[26], and questioned by Chickering [27], may object to our use of decreasing MI to choose numbers of paths to a descendant. However, we do not seek to extend our search for paths beyond the simple question of “one path or no paths?” It was this extension by Cheng, et al. which led to the inconsistencies noted by Chickering.

While we allow for some mutual information to remain after the center variable ( $V_1$  in Figure 30) is instantiated, we do not assume that there are paths remaining that provide that mutual information; rather we understand that in the data sets we use, perfect dependence or independence is unlikely.

The question arises as to what level of decrease will be considered significant when making the test for a drop in mutual information. While the threshold  $\delta=1$  is unrealistic, too low of a threshold will prevent us from finding parent-child relationships in physically correlated variables—a problem we have planned to avoid.

Our only method of determining  $\delta$  given the unknown nature of the data correlations is to test various thresholds empirically and determine a reasonable balance of first-level feature set size, stability, and error rate.

The results of just such an analysis are shown below. Using the Leukemia data set, and holding all other parameters constant, many trials at various values of  $\delta$  were performed, and the error rate and number of variables noted. As can be seen, the cross-validated error rate remained relatively constant through the reasonable range for  $\delta$ , but, as expected, the number of variables began to rise.

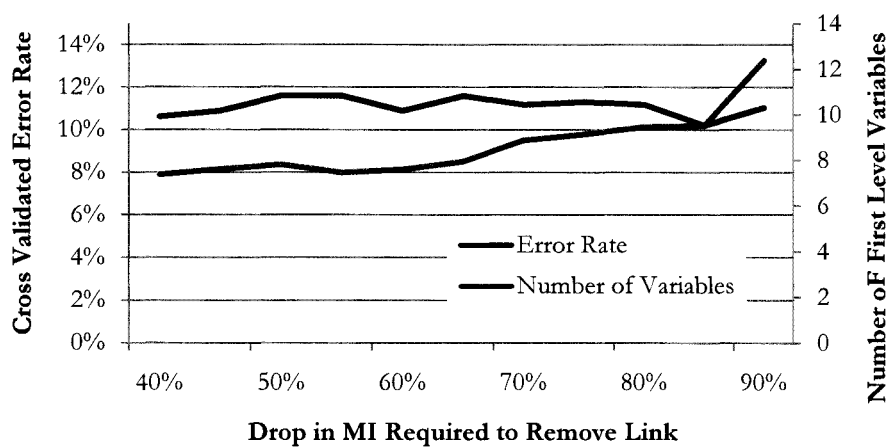


Figure 31: Effect of increasing drop threshold

The largest increase began (for this data set) at about  $\delta=0.7$ , and that threshold was chosen for the remainder of the testing. Stability of the feature set is discussed later.

Whether or not the triply-connected structure is resolved into a serial structure, all connections  $V \rightarrow V$  remaining at this point are tested for possible combination into metavariables.

### **Metavariables**

Knowing that there is a strong possibility (one that is verified later) that such  $V \rightarrow V$  connections will often represent the abundance of a single molecule showing up as more than one variable, it is prudent to attempt to recombine those variables into a single metavariable. The simple summation of abundances would seem to be the best method. Unfortunately, using a MALDI TOF-MS system, the measured abundance of molecules in a sample is made in arbitrary units, since numbers of ions are not measured directly. Even the measured abundances for two peaks in a single sample are not perfect indicators of the actual ratio of concentrations [28].

Lacking any precise method for learning the original abundance of the parent ion, we can only rely on the knowledge that the expectation value of the sum of two random variables is the sum of their individual expected values [29]. We therefore chose to sum the abundance values of two variables, if they are determined to likely represent the same parent species.

The determination of a physical relationship itself was more problematic. While it was relatively simple to identify multiply charged ionization states from the integer ratios of the two variables'  $m/z$  values, it is extremely difficult to automate the identification of the hundreds of possible adducts and modifications. In fact, this process is a separate and active area of study; a few products are available commercially.

While it may have instead been possible to group species *manually* prior to metavariable selection,<sup>24</sup> a different approach was chosen. In that approach, each candidate was combined on a trial basis, and the new variable was re-optimized to determine the maximum  $MI(C;V)$  of the metavariable. If this value exceeded the mutual information of the parent variable alone, the metavariable was kept. If the mutual information showed no increase, the metavariable was not created, and the child was noted and removed (unless it was a first level variable itself).

Surprisingly, few features were combined using this test. We had expected the series of features in the Leukemia data set around 11.7 kDa, which appeared to be a set of modifications, to combine into a single feature with increased mutual information with the class. While two features did often combine, and others did occasionally, the infrequency with which this occurred was unexpected. Table 10 below shows the occurrence of metavariable creation during ten 10-fold cross-validation trials (100 possible occurrences).

---

<sup>24</sup> This is quite difficult in itself, particularly in the mass ranges under consideration, as the exact  $m/z$  values are estimated, and there may be many possible explanations for a single  $m/z$  difference.

**Table 10: Metavariable creation in 199 feature leukemia data**

<b>First Level Feature<sup>25</sup></b>	<b>Feature Combined</b>	<b>Fraction of Occurrences</b>
42	43	0.28
43	42	0.01
76	17	0.01
76	74	0.01
100	122	0.03
121	123	0.01
122	100	0.01
122	121	0.01
145	121	0.01
145	122	0.07
151	112	0.04
198	195	0.09
198	196	0.02
198	197	0.03
198	199	0.70

In the PCA data, the occurrence of metavariable combination was more frequent. However, the much lower MI threshold for declaring relationships does not allow a direct comparison between the two data sets; at the thresholds used in the Leukemia data, the PCA data shows no connections at all.

It is clear that, if there is more information to be found by combining variables, more work needs to be done in this area. However, since our primary goal was to identify feature groups (the candidates for metavariables) rather than build a perfect classifier, we accepted the limitations of the metavariable technique as described.

## Parameter Learning

With the creation of the metavariables, the algorithm now learns the probability tables associated with each arc. This process is the same as that described in

---

<sup>25</sup> Two features may show up in reverse order in this table, such as the pair 42-43, since on various trials, either one may be selected as a parent of the other and hence be listed as the top-level feature.

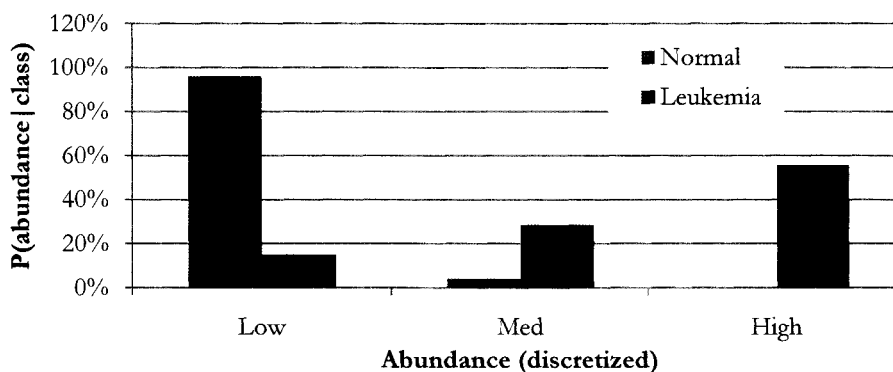
Chapter 5: Application of the Naïve Bayesian Classifier; data is simply partitioned and counted for each probability needed.

Table 11 is an example probability table from a single cross-validation attempt for the 199 feature Leukemia data. On this particular attempt, all the test cases were classified correctly, using the feature set {3,42,43,51,142,144,145,151,193,198}. The variable labeled 198 on this attempt is a metavariable combining feature 198 with features 197 and 199.

**Table 11: Probability table:  $P(\text{abundance} \mid \text{class})$  for metavariable 198**

		Abundance		
		<i>Low</i>	<i>Med</i>	<i>High</i>
Class	<i>Normal</i>	96%	4%	0%
	<i>Leukemia</i>	15%	29%	56%

Figure 32 demonstrates why this variable was chosen. It has large differences between  $P(V \mid C=\text{Normal})$  and  $P(V \mid C=\text{Leukemia})$  in the outer bins.



**Figure 32: Abundance probability differences by class**



## Classification and Error Rates

As was done in the NBC, a deterministic classification method is used. The joint probability distribution that is represented by the BN resulting from the previous steps is solved for  $P(\text{class} | \text{data})$ . The vector of data includes only those features finally chosen as first level nodes.

This probabilistic classification is matched against the known classification at some threshold, typically 0.5, and each test case is scored “correct” or “incorrect.” Once a complete set of cases is scored after one  $n$ -fold cross-validation, an overall error rate is assigned to that trial. Specific results are found in the next chapter.

## CHAPTER 7: RESULTS AND ANALYSIS

In this chapter, we present the results of the methods described in the two previous chapters applied to the three data sets described in Chapter 3. The results are organized by data set, then by method, with an overall analysis of each data set's results at the end of each section.

For each data set, two experiments are run using the Naïve Bayesian Classifier. First, ten independent trials of forward selection are done, with 20 features selected in each trial, to assess the stability of the feature set. Second, a forward selection is run, selecting a large subset of features, from which backward elimination is used to find a minimal feature set based on lowest error rate.

Thirty cross-validated error rates are calculated after each feature's addition or elimination; the results are averaged to score that feature's value to the feature set. 10-fold cross validation is used for each trial. A deterministic classification scheme is used, with a threshold of  $P(\text{class} | \text{data}) > 0.5$  to declare the class. Four and six bin discretizations are used, depending on the data.

For the Bayesian Network technique, it was first necessary to determine the appropriate thresholds for mutual information tests that create, or break, links in the Bayesian network. Node-to-node minimum MI thresholds were found first using an analysis that balanced error rate and feature set size. Next, the threshold used to remove or arrange links are determined using a similar analysis.

When those parameters were established, the algorithm was run repeatedly, typically 100 times using 10-fold stratified cross-validation. This results in 1000 Bayesian networks, and 100 fully cross-validated error rates. Networks are examined for stability and consistency, and important features are noted and compared to the NBC results.

## Generated Data

The generated data set is described in Chapter 3. The reader may want to review Table 1: *Diagnostic variables, generated data* on page 25 to see the purpose for the features listed below. We expect to find thirteen total features:

- Peak 200 as a parent, with children at 100 and 196-199;
- Peak 99 as a child of peak 199;
- Peaks 1 and 2, with correlations due to deconvolution problems;
- Peaks 3 and 4, which were independent
- Peaks 50 and 150 which should combine to form a single feature

## Naïve Bayesian Classifier

The data was processed using repeated forward selection and a “forward-then-backward” experiment.

## Forward Selection

Forward selection of features was accomplished by selecting the best 20 features (based on error rate) in ten independent trials. The goal was to select the thirteen diagnostic features, followed by several features that were not intended to be diagnostic—we will call these *random features*. During those trials, cross-validated error rate dropped to about 2%. Three typical trials are shown in Figure 33.

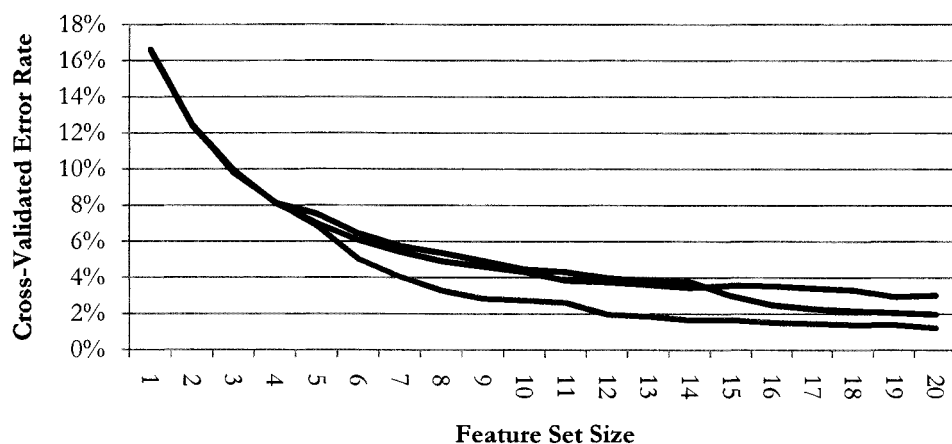


Figure 33: Error rate during forward selection, generated data

Table 12 lists the features selected, in order, for each of the ten trials. The feature set selected early in each trial was relatively stable. The first three selections were consistent, and included the parent peak (200) of the correlated set {200,196-199, 99-100}, followed by one of the correlated “convoluted peaks,” feature 2. The third selection in every trial was one of the pair of correlated “fragments,” peak 50. The fourth selection in all trials was a random peak, the fifth was often a “mildly diagnostic” feature 3. The other mildly diagnostic peak, feature 4, does not appear. Entries in red are those intended to be diagnostic.

Table 12: Forward selection, generated data

Selection No.	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10
1	200	200	200	200	200	200	200	200	200	200
2	2	2	2	2	2	2	2	2	2	2
3	50	50	50	50	50	50	50	50	50	50
4	179	190	190	5	190	179	190	190	190	190
5	1	3	61	94	3	170	3	3	3	3
6	175	164	81	170	164	62	164	164	164	164
7	58	126	94	195	76	81	126	76	76	126
8	3	76	112	190	126	5	76	126	126	76
9	48	63	193	37	63	36	63	63	63	63
10	194	94	170	61	160	76	138	160	178	160

It would have been difficult to separate the mildly diagnostic feature 3 from the random features such as 190 that appeared frequently in the first few selections. It is clear, however, that the error rate curves for the ten trials begins to diverge at selection 4, which could indicate that a maximum-size stable feature set has been selected. The error rates for all ten trials are in Appendix C.

### **Forward-Backward Feature Set and Error Rates**

In the second experiment, 80 of 100 features are selected by forward selection, and then backward elimination is run using only those features. Cross-validated error rates as low as 1% were obtained during trials of the forward selection portion. However, error rates under 10% were only achieved while selecting many random features<sup>26</sup>. Of the thirteen diagnostic features, only 5 were chosen. Given the problems associated with correlated features, we expected only one of the correlated set {200,199-196,99-100} to be chosen; that expectation held true. Of the six remaining features, namely {1,2,3,4,50,150}, four were selected; 2 were not. Table 13 shows the diagnostic peaks selected and the selection order (of 80). Features highlighted in red were found by forward selection in the first experiment.

---

<sup>26</sup> More specifically, “features that were diagnostic only due to random chance in a small sample set.”

Table 13: Diagnostic variables selected, generated data

Feature	Selection Number
200	1
1	2
3	4
2	26
150	69

Cross-validated error rates for the forward selection portion of this experiment stabilized at about 1% with 25 features remaining. The error rate curve is shown in Figure 34.

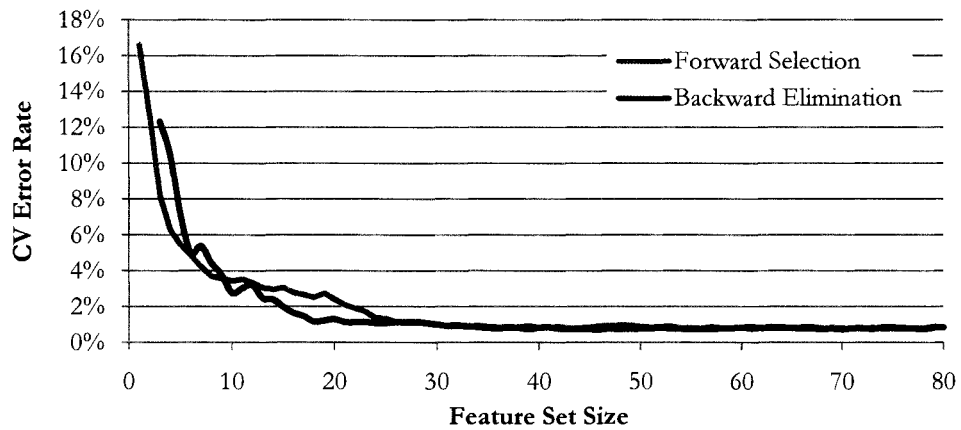


Figure 34: Error rate during feature selection, generated data

As the error rate began to rise during the backward elimination portion of this experiment (the red line in Figure 34 above, reading right to left), all the features listed above remained, as well as 13 random features. After further elimination, feature 2 was removed, leaving 4 diagnostic and 12 random features. Feature 2 was highly correlated to feature 1 due to artificial deconvolution problems. After this, however, all the random features were eliminated, leaving four diagnostic features, namely {200, 1, 3, 150}, with about 7% error.

This feature set includes exactly one feature from each group of correlated features, other than feature 4, which should have been independent and diagnostic. A closer examination of feature 4 shows that its distribution is perhaps less diagnostic than was intended. Figure 35 shows features 3 and 4 side-by-side for comparison.

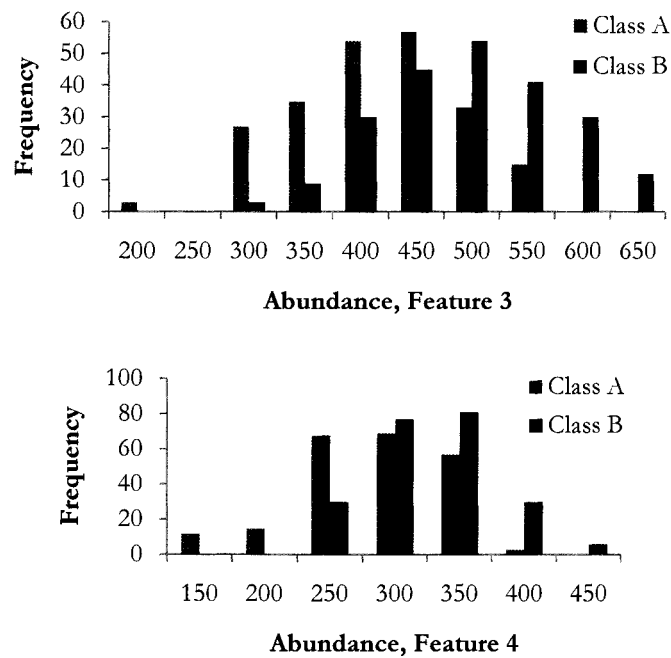


Figure 35: Distribution of features 3 and 4, generated data

### Bayesian Network

The artificially generated data set was processed using the Bayesian network algorithm described in Chapter 6; the code is in Appendix B: MATLAB Code. Prior to the main part of the experiment, it was necessary to determine the thresholds for mutual information tests, as described on page 82.

It was immediately clear that thresholds for  $MI(\text{class};\text{variable})$  near the baseline, which was derived from the expected maximum MI between a similar size data set with a random class, overstated the number of diagnostic features. Even using 2.0 times the baseline as a threshold, 16 features were frequently selected as diagnostic. Knowing that we intentionally placed only 13 of 200 diagnostic features in the data set, it was obvious that small sample effects<sup>27</sup> allowed some random features to appear non-random, when optimized as described on page 85. Figure 36 shows the effect of changing the threshold on error rate and number of variables selected.

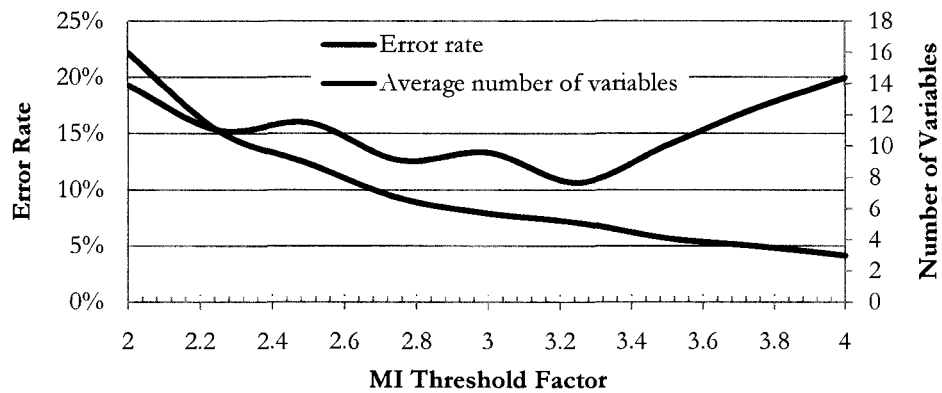


Figure 36: Effect of MI Threshold

A MI threshold factor of 3.2 times a “random MI” minimized error rate and limited feature set size to about 6. While this was less than the 13 known diagnostic variables, we expected that the correlated variables would not be selected at the first level, so our expected feature set size was seven. We chose a somewhat lower threshold factor of 2.5, knowing that random features might be

<sup>27</sup> Meaning that our sample size of 150 samples was small compared to the 200 variables.



selected, but attempting to determine whether our methodology would identify them through their infrequent and unstable inclusion.

It was also necessary to determine the threshold for declaring parent-child relationships in the network structure. An analysis similar to that described on page 88 (parent-child identification) determined that a 75% drop from  $MI(\text{class}; \text{child})$  to  $MI(\text{class}; \text{child}|\text{parent})$ , where “child” is the perspective child variable of the “parent” variable, is a reasonable threshold. This threshold provided a minimum error rate (about 5% in that analysis) while maintaining the feature set size that was desired. The remainder of the analysis was performed with these two thresholds.

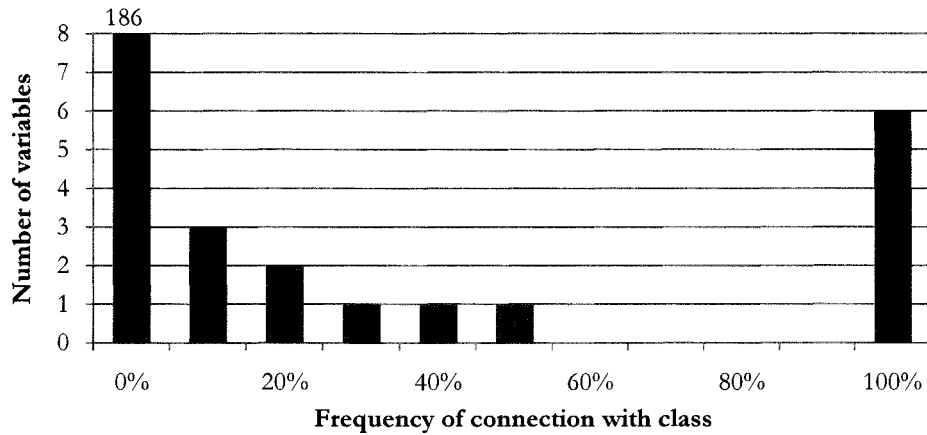
### **Feature Set Selection**

We performed 100 repetitions of the Bayesian network algorithm, each time recording the adjacency matrices and 10-fold cross-validated error rate, as well as any metavariables created. Since each cross-validation attempt results in a unique Bayesian network and accompanying parameters, with 100 repetitions of 10-fold cross-validation, 1000 adjacency matrices are created.

#### Class to feature connections

The adjacency matrices for all trials are summed together; the value at the  $(i,j)$  position of the result is the total number of times a connection was found from  $V_i$  to  $V_j$  in all networks. Since the class variable is a node in the network, the final row in the adjacency matrix represents the number of times a connection was found

from  $C \rightarrow V$  for each variable. Figure 37 shows how many of the 200 variables were connected to the class at various fractions of the possible networks.



**Figure 37: Frequency of class-variable connections, generated data**

Six features are selected by the BN algorithm to be directly connected to the class more than 50% of the time – in fact, these six are always selected. These features are listed in Table 14. Features that were also found in the NBC are highlighted in red.

**Table 14: Variables selected by BN, generated data**

Feature	Selection Frequency
<b>1</b>	100%
<b>2</b>	100%
<b>3</b>	95.1%
<b>4</b>	99.7%
<b>150</b>	99.9%
<b>200</b>	99.6%

This list is almost exactly the set of features we expected to find. It includes all the “parent” features we attempted to place in the generated data set, with none of the correlated features. The seven features not included were:

- 50, which, with 150, was a fragment of a hidden feature;
- 99 and 100, which were ionization states of 199 and 200; and
- 196-199, which were correlated modifications of 200.

Feature 99 was found to be a first level node nearly 40% of the time; a random feature (155) was included in 48% of the trials. We next examined the second level features to determine whether these correlated features were identified.

#### Feature to feature connections

Only two first-level features had other features frequently connected at the second level. Feature 200 was found to be the parent of features 195-199 and 100, all more than 99% of the trials. Feature 150 was connected to feature 50, but in only 13% of the trials. These two features were fragments of a non-measured feature.

The only diagnostic feature not identified by the BN algorithm at either the first level or the second level of nodes more than 50% of the time was feature 99. However, this was a correct result, since feature 99 was intended to be a child of feature 199, which itself was derived from feature 200. Therefore, it should have been identified as a third level node and eliminated, which is indeed what occurred.

#### Metavariables

All the children of feature 200 were occasionally combined into a metavariable with that feature, at rates ranging from 17% (feature 100) to 31% (feature 196). Features 50 and 150 were not found to combine into a metavariable.

## Error Rates

Error rates ranged from 10% to 19%, much higher than that found with the NBC.

A histogram of cross-validated error rates for 100 trials is presented in Figure 38.

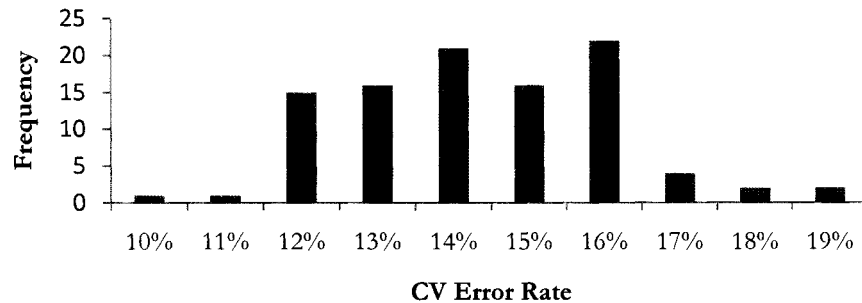


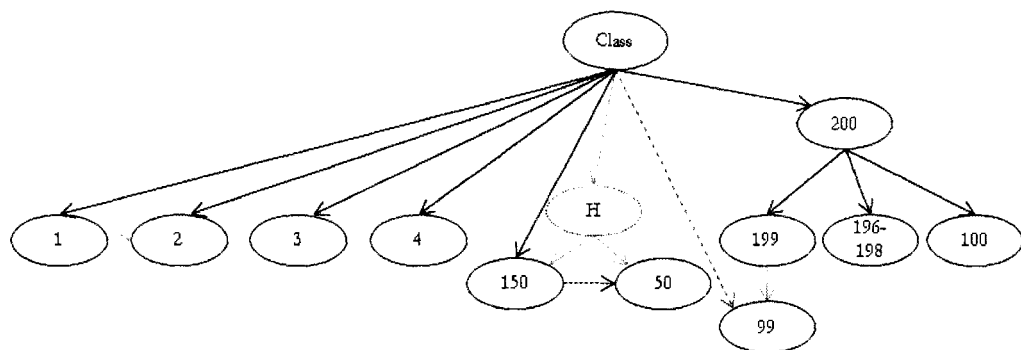
Figure 38: Error rate distribution, generated data

## Analysis

The Naïve Bayesian Classifier was extremely successful at minimizing error rate, achieving cross-validated rates as low as 1% with feature set sizes near 20. It did so, however, by including large numbers of random features and missing many diagnostic features—both primary and correlated. We did see some stability in the first several features selected, but large instabilities in the feature sets that achieved the lowest error rates. Correlated features, as expected, were not added to the feature sets by the NBC. Given that the lowest error rates were achieved with a large and unstable set of features, we would expect the resulting classifier to have much higher error rates for new cases.

The Bayesian network algorithm behaved much differently. It was able completely identify the expected feature set, including correlated features at the second level, and remove all non-diagnostic features and third-level features. It proved extremely

stable with regard to feature set selection. Error rates were much worse than the NBC, but likely more indicative of the true classification ability of this data. Figure 39 shows the most likely Bayesian network found by averaging the results of the 1000 trials.<sup>28</sup> Solid lines represent strong links, dotted lines represent weaker links. Black lines were found by the algorithm, blue lines were intended by the construction of the generated data (see page 22) but not found. Red lines represent links that were found, but not intended.



**Figure 39: Resulting Bayesian network, generated data**

The node labeled “H” was a hidden variable; it presented itself as a serial connection between its parent (the class) and one of its descendants. Feature 99 was a third-level variable and should have been removed, but was found in a small fraction of the trials.

This result shows that the BN was successful at finding nearly all the intended links in the data, and most importantly, in showing their causal connections, something that the NBC was unable to do.

<sup>28</sup> The 1000 adjacency matrices are averaged; arcs that appear frequently are declared “stable.” The resulting network could be used for classification of new samples if needed.

## Leukemia Data

Portions of the results shown in this section are shown elsewhere in the document; they are repeated here for consistency. The 199-feature data set is used for these calculations.

### Naïve Bayesian Classifier

As was done for the generated data, a repeated trials of forward selection investigates stability, and a “forward-then-backward” trial seeks a minimal feature set. The threshold for deterministically declaring a class remains at  $P(\text{class} | \text{data})=0.50$ , and 10-fold cross validation is used again.

### Forward Selection

As in the generated data, 20 features are selected in ten trials. However, the number of diagnostic features is now unknown. Three typical error rate profiles are shown in Figure 40.

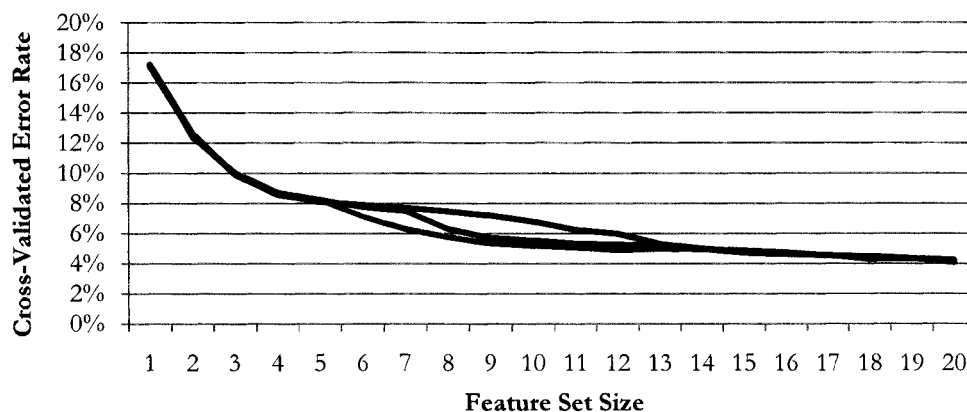


Figure 40: Error rate during forward selection, Leukemia data

The following features were found to occur at least 75% of the trials:

<b>Features</b>	<b>Frequency</b>
<b>199</b>	100% of trials
<b>198</b>	
<b>151</b>	
<b>122</b>	90% of trials
<b>141</b>	
<b>193</b>	
<b>68</b>	80% of trials

Four more features, 3, 10, 38, and 43, appeared in at least one-half of the trials.

The complete list of features selected during this experiment is found in Appendix C: Results. An excerpt is shown in Table 15. Features mentioned above are listed in red.

**Table 15: Forward selection, leukemia data**

<b>Selection</b>	<b>Trial 1</b>	<b>Trial 2</b>	<b>Trial 3</b>	<b>Trial 4</b>	<b>Trial 5</b>	<b>Trial 6</b>	<b>Trial 7</b>	<b>Trial 8</b>	<b>Trial 9</b>	<b>Trial 10</b>
<b>1</b>	<b>199</b>	<b>199</b>	<b>199</b>	<b>199</b>	<b>199</b>	<b>199</b>	<b>199</b>	<b>199</b>	<b>199</b>	<b>199</b>
<b>2</b>	<b>198</b>	<b>198</b>	<b>198</b>	<b>198</b>	<b>198</b>	<b>198</b>	<b>198</b>	<b>198</b>	<b>198</b>	<b>198</b>
<b>3</b>	<b>141</b>	<b>141</b>	<b>141</b>	<b>141</b>	<b>141</b>	<b>193</b>	<b>141</b>	<b>141</b>	<b>141</b>	<b>141</b>
<b>4</b>	<b>43</b>	<b>43</b>	<b>43</b>	<b>43</b>	<b>122</b>	18	80	<b>122</b>	<b>43</b>	<b>122</b>
<b>5</b>	31	<b>122</b>	4	4	9	93	<b>122</b>	9	4	9

### **Backward Elimination**

Since the size of the “true” feature set is unknown, a trial of backward elimination was completed to look for an optimal feature set size, as measured by error rate. Figure 17, which is repeated below, shows that the cross-validated error rate stops decreasing at about 6%, at that point a feature set of about 60 features remains. This experiment was repeated several times with consistent results.

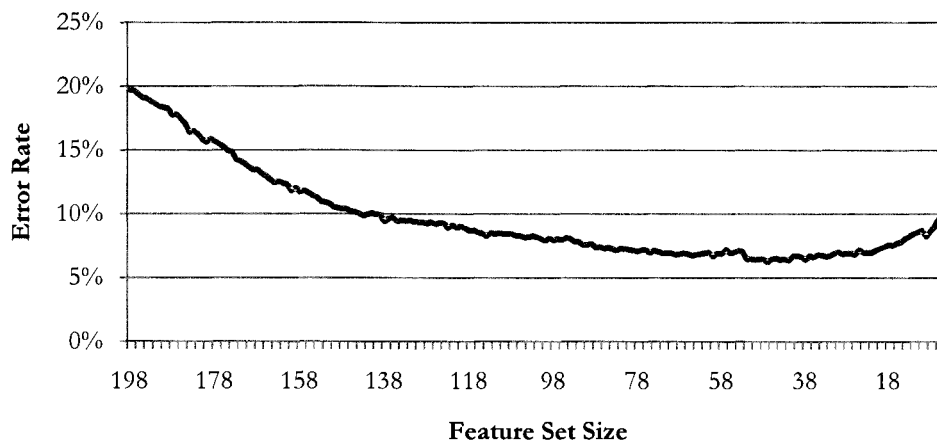


Figure 17: Error rate during backward elimination, Leukemia data

### Forward-Backward Feature Set and Error Rates

The initial portion of this experiment consisted of choosing a reduced feature set of 80 features. That size was selected to add a margin to the optimum feature set size of 60 found in the previous experiment.

Forward selection produced error rates of about 6.4% at a feature set size of 50 features and then stabilized. This error rate profile is represented by the blue curve in Figure 41.

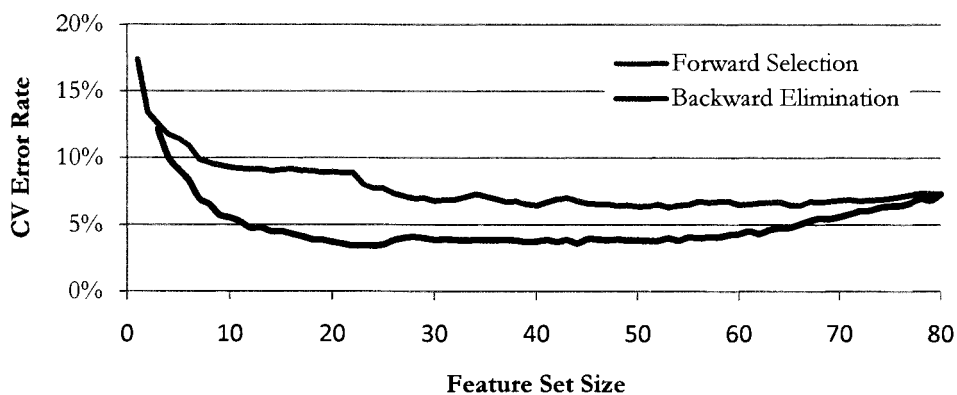


Figure 41: Error rate during feature selection, Leukemia data



Using the reduced feature set found in the forward selection phase, backward elimination was performed. This error rate profile is represented by the red line in Figure 41. Unlike the generated data, in which the backward elimination error rate closely followed the forward selection curve, in this data the error rate dropped further, reaching minimum of 3.44% with 21 features remaining. Of the 11 features found often in the forward selection trial, 7 remained in this experiment (red in Table 16). The list is in order of effect on error rate.

**Table 16: Features remaining after forward-backward selection**

Feature Label
198
193
199
144
122
182
140
151
120
117
34
124
101
186
3
115
45
38

Features 141 and 68, which were found often in forward selection but not in this particular trial, were often included in the minimal feature set during other trials. With only the seven features found in both experiments, the Leukemia data set (after normalization) can be classified at either a nominal or a cross-validated error rate of about 7.5%.

## Bayesian Network

The leukemia data set was processed by the same method described for the generated data. Prior to the main part of the experiment, it was necessary to determine the thresholds for mutual information tests, as described on page 82. As can be seen from Figure 25, reproduced below, using a MI threshold of 3.2 times the maximum random  $MI(C;V)$  achieved a minimal error rate with approximately 7-10 independent features.

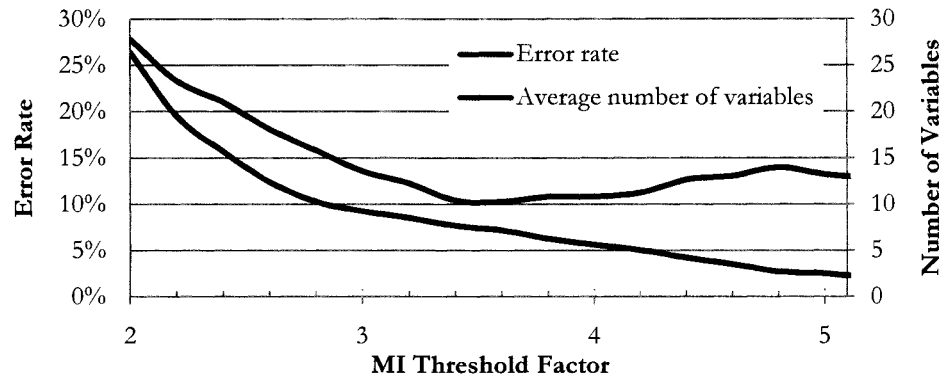


Figure 25: MI threshold effects under 10-fold cross-validation

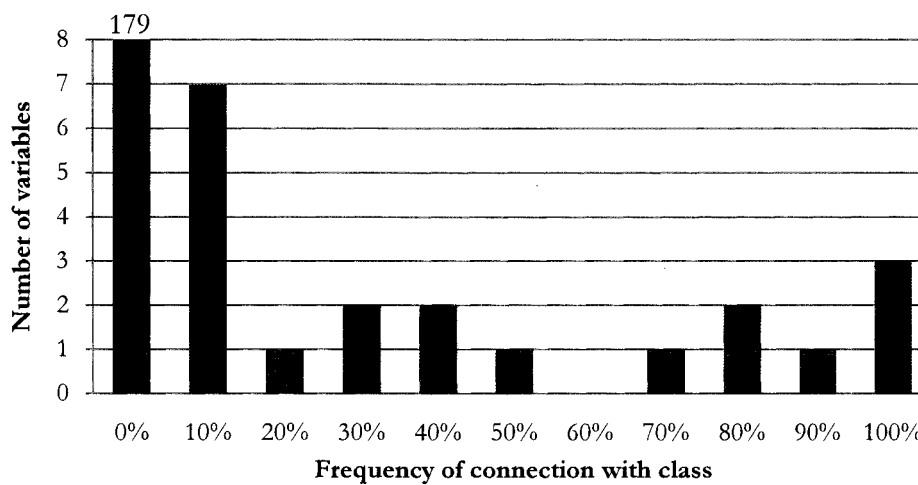
We also found that using a drop in MI of 70-80% to determine parent-child relationships provided minimal error rates and similar feature set sizes (see Figure 31, page 90). Therefore, analysis of the leukemia data set was performed with these two parameters.

## Feature Set Selection

We again performed 100 repetitions of the Bayesian network algorithm, each time recording the adjacency matrices and 10-fold cross-validated error rate, as well as any metavariables created.

### Class to feature connections

Figure 42 shows the number of variables selected at various fractions of the possible trials. The bar at the far left, representing the group of variables that were never selected, is off the scale of this chart at 179 of the 199 total variables.



**Figure 42: Frequency of class-variable connections, Leukemia data**

Twenty features are selected at least once, but only eight of these are selected more than 50% of the time. Seven features are selected by the BN algorithm to be directly connected to the class more than 75% of the time.

The eight most-selected features are listed in Table 17. Features that were also found frequently in the NBC trials are highlighted in red.

Table 17: Variables selected by BN, Leukemia data

Feature	Selection Frequency	<i>m/z</i>
<b>198</b>	100%	11697
<b>51</b>	98.8%	4106
<b>145</b>	98.0%	7691
<b>151</b>	87.8%	7863
<b>142</b>	87.4%	7483
<b>3</b>	82.3%	2798
<b>144</b>	76.0%	7649
<b>141</b>	54.6%	7448

The three features not highlighted, along with feature 141 (which was found by NBC) are all within 250 Daltons, a range which we have found to be indicative of modifications of a single protein. These features were apparently interconnected in the network; the feature selected to connect directly to the class appeared to vary as cross-validation chooses different subsets of cases from which tests are derived.

Feature-feature connections

Extremely strong and frequent second level connections are found between feature 198 and several others, particularly those between 195 and 199. The strongest connections with this feature are listed in Table 18.

Table 18: Second level features connected to feature 198

Feature	Selection Frequency	<i>m/z</i>
<b>107</b>	100%	5878
<b>108</b>	100%	5887
<b>195</b>	100%	11486
<b>196</b>	100%	11539
<b>197</b>	100%	11640
<b>199</b>	100%	11742
<b>144</b>	96.5%	7649
<b>42</b>	85.5%	3898

The features labeled 107 and 108 are almost exactly one-half of the  $m/z$  ratio of feature 199, which we have taken to mean that they are each a doubly-charged satellite of one of the features 195-199. Feature 42 is almost exactly one-third of the  $m/z$  value of feature 198, we have taken that to be a triply-charged satellite.

We are less sure about the causal connection between features 144 and 198. It may be that 144 is a fragment of 198, indeed, the difference between their  $m/z$  values is nearly equal to the  $m/z$  of feature 51, which was found to be connected to the class variable.

Feature 145 is connected to feature 146 in 78% of the trials; the difference in their  $m/z$  values is 21 Daltons, which may indicate a sodium adduct (23 Daltons) or neutral loss of water (18 Daltons). It is connected to feature 122 less often (33%).

Features 3, 142, 144, and 151 have no frequent second level connections with other variables. Due to the  $m/z$  proximity of the features in the range 141-145, it may be that a single diagnostic feature is in this range and being modified. It is also possible that there is a larger protein outside the range we have studied which is showing up here as multiply charged states or fragments. We were unable to determine specific patterns for these features, but are currently attempting to expand our mass range to find more massive features correlated to this group.

Feature 38, which was found in the NBC, is not found in the first level connections of the BN. However, it is found to be connected at the second level with feature 141, and appears to be a doubly-charged ionization state.

### Metavariables

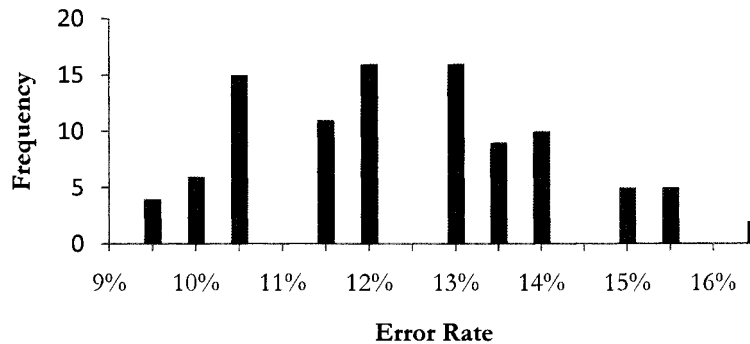
The algorithm attempts to recombine these feature combinations to create metavariables with better classification performance.

All the features 195, 196, 197, and 199 are combined at least once with parent feature 198 during this trial, however, only 195 (65.5%) and 199 (47.1%) are frequently combined.

Feature 122, which was one of the features found often by the NBC, was combined with feature 145 nearly 20% of trials. It does not appear to be an adduct, modification, or satellite of 145.

### **Error Rates**

Cross-validated error rates averaged 12.2%, significantly higher than the NBC. However, the feature sets were much smaller (averaging  $9 \pm 2$  features) and much more stable. The range of error-rates is shown as a histogram in Figure 43.



**Figure 43: Histogram of CV error rates, Leukemia data**

Only 100 values are included, as each 10-fold cross-validation results in a single error rate for the entire population.

### Analysis

Many of the results found in the Leukemia data set were similar to those predicted with the generated data (which was designed to mimic it). We saw very low cross-validated error rates (4%) and even lower nominal error rates (2.5%), but this required feature set sizes near 20. Features selected by the NBC were very unstable past the selection of a few features, and correlations were ignored.

The BN algorithm identified the most stable features found by the NBC, as well as other stable features. The result of that analysis is shown in Figure 44.

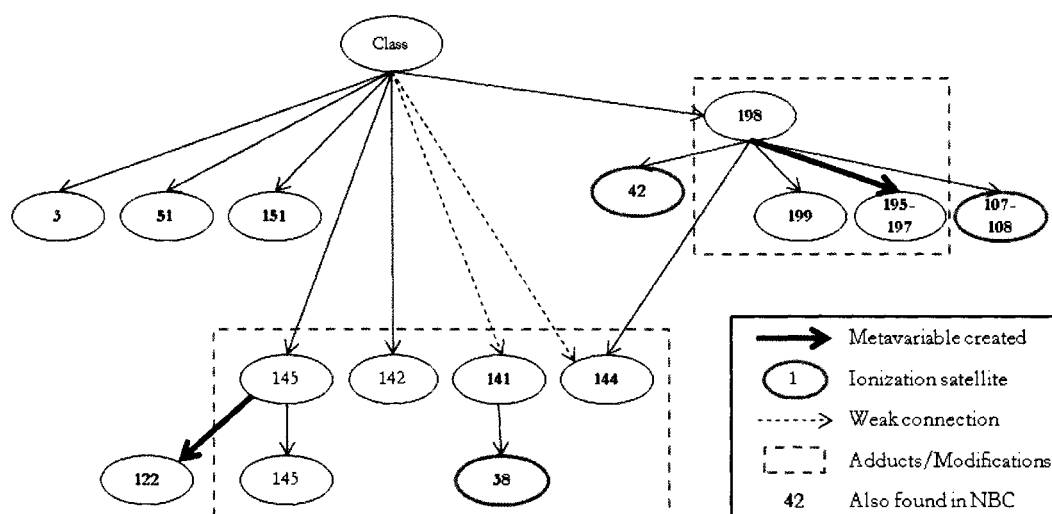


Figure 44: Resulting Bayesian network, Leukemia data

This stability came at the cost of a rise in cross-validated error rate, however. The BN also successfully found and connected correlated features such as

modifications and ionization states. One of these feature groups has possibly led us to a better biomarker candidate than had been found using previous methods.

Our collaborators at EVMS had previously identified the feature labeled here as 198 as a potential biomarker [8]. However, attempts to identify the actual protein consistently have been problematic [2].

The additional understanding of correlation given by the BN analysis, however, gave us important clues that may lead to better protein identification. The right half of the structure shown in red in Figure 45 is the one that produces the features 195-199 in our data set. The normal (black) spectrum has a peak located at the position we label as feature 198, but typically at a much lower abundance, and without the modifications to feature 198 seen in the disease (red) spectrum.

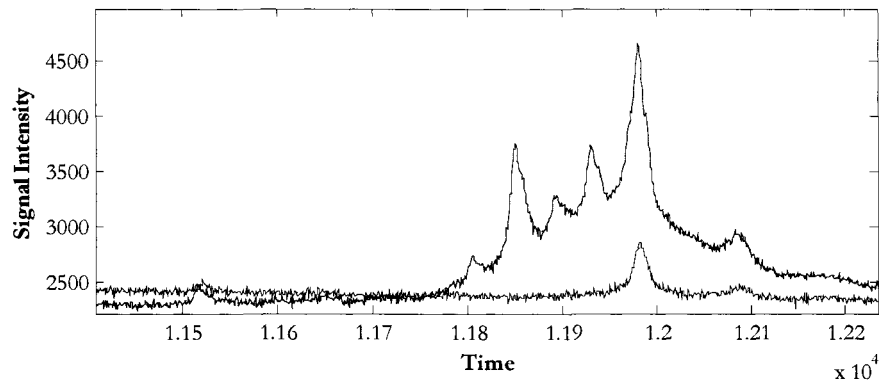


Figure 45: Leukemia (red) and normal spectra, vicinity feature 198

Examination of the  $m/z$  differences between the parent feature (198) and the others has led us to hypothesize that this feature represents the protein serum amyloid A (SAA) with modifications such as des-arginine (156 Daltons) which matched our  $m/z$  change from feature 198 to feature 196 (157 Daltons) [30]. This



combination of SAA and modifications has been found to be present in the blood sera of renal cancer patients, also using a SELDI technique [31]. Further work to confirm this identification is currently ongoing at EVMS.

## Prostate Cancer Data

Examination of the PCA data mirrored that of the other data sets. However, we found that feature set selection was more difficult than the Leukemia data, and error rates were much higher.

### Naïve Bayesian Classifier

Due to the difficulties with stable feature set selection, we examined error rate curves for both the forward selection and backward elimination methods to determine the approximate optimal feature set size for the PCA data. We then used many repetitions of forward selection to determine which features were consistently chosen within this parameter.

### Forward Selection

Several trials of forward selection of nearly all the variables produced error rate curves such as that in Figure 46.

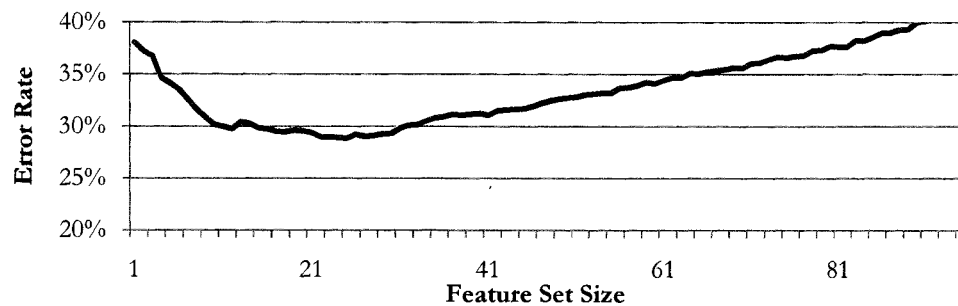


Figure 46: Error rate during forward selection, PCA data

The minimum error rate is just under 30%, far greater than in the previous data sets. This error rate is achieved at about 22 features. We will therefore expand the repetitive forward selection feature set search to 30 features in an attempt to find a stable feature set of about 20 features.

After 10 trials of selecting 30 features (the complete list is in the appendix) there are 23 features that appear in more than one-half of the feature sets. The error rate is more unstable initially than seen in previous data sets. Three typical error rate curves are shown in Figure 47. The minimum error rate was 28%.

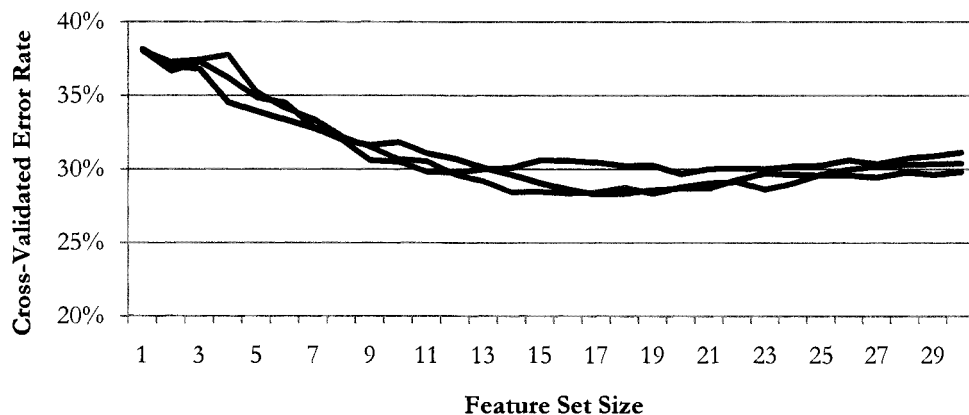
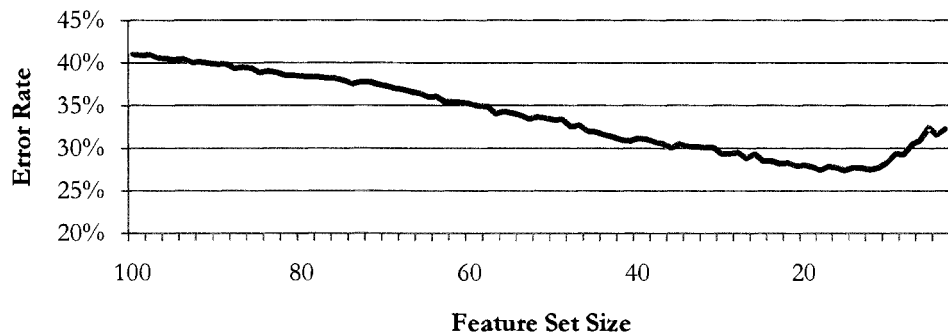


Figure 47: Error rate during repeated forward selection, PCA data

### Backward Elimination

Backward elimination achieved a slightly lower error rate (27%) at only 12 features when compared to forward selection, which achieved a minimum error rate of 28% at about 22 features.



**Figure 48: Error rate during backward elimination, PCA data**

The 12 features comprising the minimum error rate feature set in this trial are shown in Table 19. Those that were also found often during the repeated forward selection trials are highlighted in red.

**Table 19: Features achieving minimum error, backward elimination**

<u>Feature Label</u>
38
45
19
40
52
58
84
23
89
66
67

### **Forward-Backward Feature Set and Error Rates**

During forward selection, 60 of the 100 possible features were chosen to provide a reduced feature set. As before, minimal error rates were achieved at about 22 features.

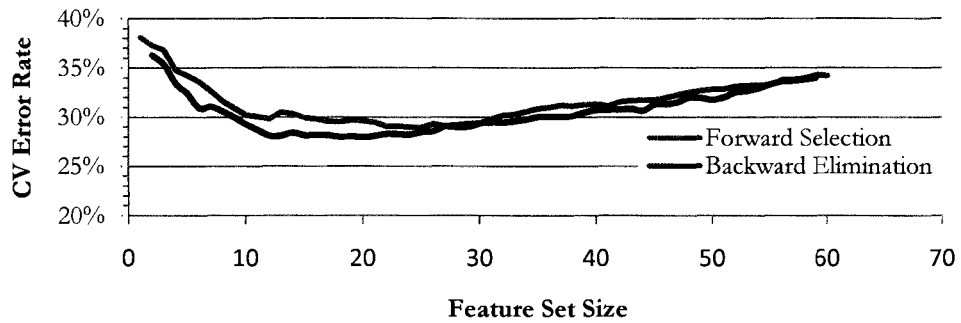


Figure 49: Error rate during feature selection, PCA data

Backward elimination is then attempted to find a minimal feature set; Figure 49 shows the error rate profile for both phases. Error rates dropped to 28% at 22 features and remain steady until another 10 features are eliminated. The final set of features is listed in Table 20. Features found often in repeated forward selection are listed in red.

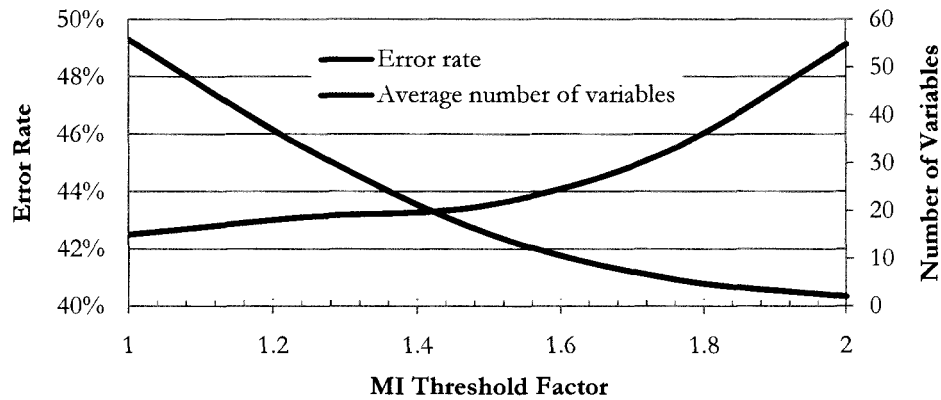
Table 20: Minimal feature set, forward-backward selection, PCA data

Feature Label
7
23
75
9
37
58
38
73
64
77
25

### Bayesian Network

As was done for the Leukemia data, we examine a range of mutual information thresholds used to declare node connections in the Bayesian network. The thresholds that give reasonable results in the PCA data set are much closer to the

baseline value of one, representing the maximal value of mutual information between similar data and a random class.



**Figure 50: MI threshold effects under 10-fold cross-validation**

Figure 50 shows that minimal cross-validated error rates are achieved with a threshold factor of 1.0 to 1.5, while the number of first-level features decreases from 55 to 3 over the range 1.0 to 1.8. We chose to use a factor of 1.5, in order to expand slightly the minimum feature set size with the goal of finding stable (and unstable) features.

Determining the proper mutual information drop threshold was more difficult. As shown in Figure 51, the feature set size (and to some extent, error rate) were unstable while varying this parameter. Since the feature set size was unknown, and nothing in Figure 51 suggested otherwise, we chose to use 70% as a parameter to minimize error rate.

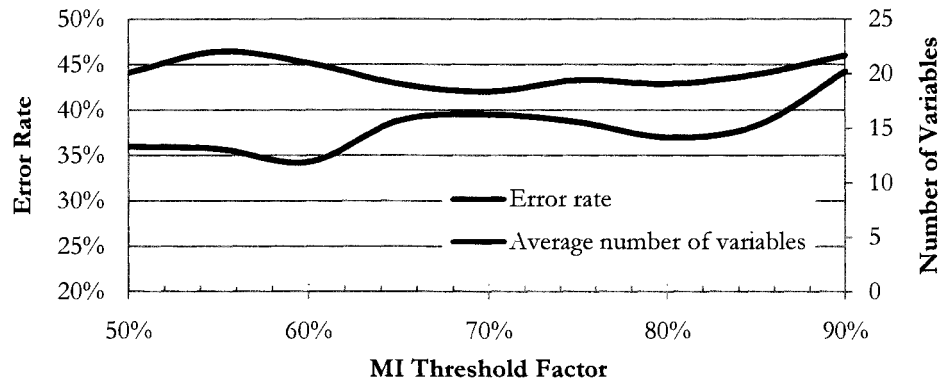


Figure 51: Effect of increasing drop threshold, PCA data

### Feature Set Selection

As was done for the Leukemia data, 1000 networks were created by repeating a 10-fold cross validation 100 times.

#### Class to variable connections

During those trials, 16 features were found in more than 30% of the trials. Those features and their selection frequency are found in Table 21. Features found frequently by the NBC are highlighted in red. This data set did not exhibit the feature set consistency shown by the previous two data sets between the NBC and the BN approaches. Of the 9 features found in more than 50% of the BN trials as first level features, only 4 are from the two sets of about 12 features in the previous sections.

Table 21: Variables selected by BN, PCA data

Feature	Selection Frequency	Mass/charge
23	87.4%	4042
49	72.2%	7406
67	71.7%	8779
25	69.6%	4083
79	63.6%	13260
85	62.6%	16580
74	59.0%	9672
99	57.1%	81441
89	53.8%	25091
45	48.5%	6607
70	46.8%	9256
33	45.5%	4553
66	44.9%	8740
77	42.1%	12555
65	37.7%	8656
62	37.2%	8179

Feature-feature connections

No features were found to connect to the most frequently selected first level feature, labeled 23. The next most frequent, feature 49, had several second level connections, including  $m/z$  neighbors 46, 47, 50, and 51. A list of the various second level connections for the most frequently found first-level features (first row) is found in Table 22, with NBC-identified features in red. We were unable to make specific hypotheses about the nature of the connections listed above, other than those features which are adjacent and may be adducts or modifications.

Table 22: Variables connected to first-level variables, PCA data

23	25	33	45	49	62	65	66	67	70	74	77	79	85	89	99
	2			46				44		<b>73</b>		60	88		
	13			47				50							
	24			50				63							
	<b>37</b>			51				65							
								68							
								69							
								<b>71</b>							
								72							
								95							

We do have indications that some of the relationships shown are real; feature 67 may be Apolipoprotein C II, which has correlated modifications that may be represented by features 68, 69, 71, and 72. We continue to investigate this finding.

#### Metavariables

Feature 44 and 67 were combined into a metavariable in 53% of the trials. Feature 66 was also combined with feature 67 in about of 30% of the trials in which it appeared as a first level variable.

#### **Error Rates**

Error rates for this data set were poor, averaging 44% and never dropping below 39% for any of the 100 cross-validated trials. A histogram of the cross-validated error rates that occurred in this set of trials is presented in Figure 52.



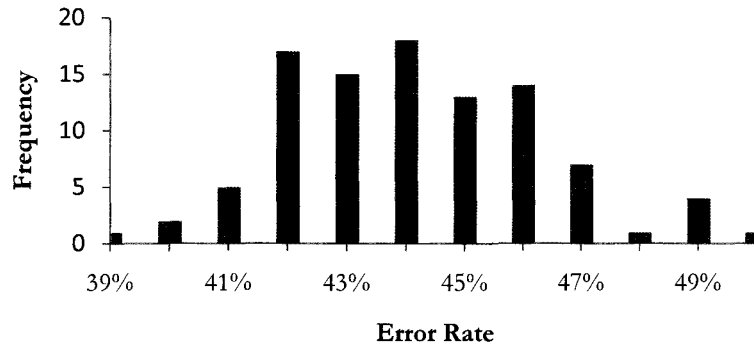


Figure 52: Error rates from the BN algorithm, PCA data

### Analysis

Neither the naïve Bayesian classifier, nor the Bayesian network approach, was successful in finding either stable feature sets or low error rates. A small number of features did appear as diagnostic in several different experiments, but the extremely high error rates, and unstable inclusion of other features in the results, led us to conclude that these techniques are unlikely to identify any features as likely biomarker candidates. Further research, particularly in the signal processing stage (peak-picking, background subtraction, etc.) is ongoing and may help improve these results.

## CHAPTER 8: CONCLUSION

Biomarker discovery via mass spectrometry of biologic samples has been an intense area of recent research. The possibility of semi-automated high-throughput, multiple-disease tests for deadly cancers is enticing, but the systemic errors in the mass spectrometry signals have led to generally poor results to date.

Proteins with relatively high mass-to-charge ratios have been found using traditional data analysis techniques, such as the 11.7 kDa peak in our Leukemia data set. Features such as that peak, seen in Figure 53 below, are identified by relatively simple statistical tests.

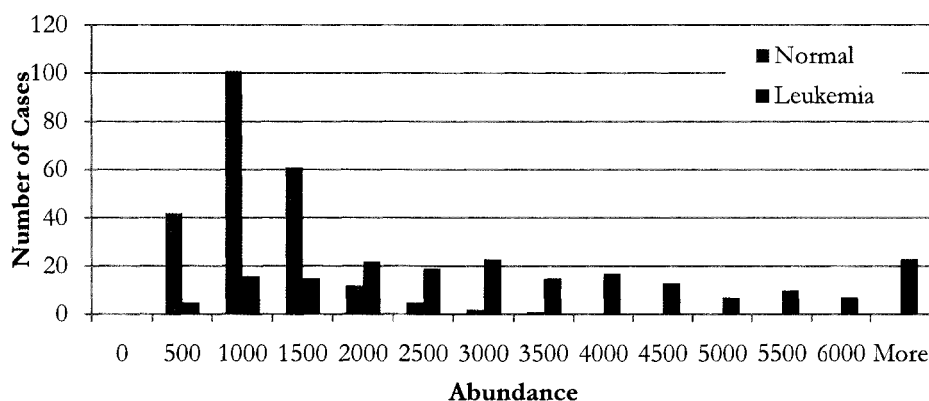


Figure 53: Distribution of 11.7 kDa peak by class, Leukemia data

Even after finding such a diagnostic feature, however, the task of identifying the exact protein can be difficult—as our group has discovered. Our naïve Bayesian classifier, combined with stratified  $n$ -fold cross-validation, expanded our ability to find feature sets with modest stability, but did not solve the problem of identifying correlated features. The Bayesian network classifier appears to perform this

function well in some data, and appears to identify the causal relations between correlated features. This capability has already led us to re-designate the protein responsible for the primary Leukemia feature shown in Figure 53 above as serum amyloid A (with modifications), which appears to be consistent with findings of other researchers' results.

The fact that the BN classifier was unsuccessful in performing the same function with the prostate cancer data is somewhat encouraging, in that it shows a certain degree of discrimination between “good” and “bad” data. High-dimensional data with small samples, such as the data sets in this study, can often lead traditional statistical tests into identifying correlations that will not be stable under new cases. The combination of our algorithms provide a method of attacking the problem from two very different angles – the wrapper approach of the NBC, based on error rates, and the filter approach of the BN, which ignores error rates and instead focuses on the model-free mutual information score. This combination allowed us to determine that the current state of the prostate cancer data did not lead to a stable feature set that could be exploited for biologic information.

We will continue to explore ways of increasing the repeatability of our abundance measurements in the pre-processing stages. Whether that effort is successful or not, the naïve Bayesian classifier, and mutual information-based Bayesian network algorithm, will be important tools in the search for biomarkers.

## APPENDIX A: MATHEMATICS

### Maximum Entropy

Entropy is defined as

$$H(X) = - \sum_x P(x) \log_2 P(x). \quad (\text{A-1})$$

We can find the maximum entropy using the method of Lagrange multipliers, adding the constraint  $\sum_x P(x) = 1$  (the sum of all probabilities must be 1). In both summations, the value of  $x$  ranges over all possible values that the corresponding variable  $X$  can take. Labeling these values  $\{x_1, x_2, \dots, x_n\}$  and using the shorthand notation  $p_i = P(x_i)$ , the constrained equation is

$$H(X) = - \sum_i p_i \log p_i + \lambda \left[ \sum_j p_j - 1 \right]. \quad (\text{A-2})$$

Taking the partial derivative of the above with respect to one of the particular probabilities  $p_m$ , with  $\partial p_i / \partial p_m = \delta_{im}$  (the Kronecker Delta), and setting the result to zero to find the maximized constrained solution yields

$$\frac{\partial H}{\partial p_m} = - \log p_m - 1 + \lambda = 0. \quad (\text{A-3})$$

Since  $\lambda$  is a constant, and equation is valid for all  $m$ , this implies all the probabilities are equal to some other constant  $\gamma \equiv e^{-\lambda-1}$ . Applying the constraint  $\sum_{i=1}^n \gamma = 1$ , we find that all probabilities  $P(X=x_i) = 1/n$ , where  $n$  is the number of possible values that  $X$  can take.

Therefore, the maximum value of entropy is when all values of the variable are equally probable, or, alternatively, occur an equal number of times in a sample set.

## Maximum Mutual Information

We seek to find the maximum value of  $MI(X;Y)$ , where  $X$  and  $Y$  are discrete variables that can take on the values  $x \in \{x_1, x_2, \dots, x_n\}$  and  $y \in \{y_1, y_2, \dots, y_m\}$ , with  $m \leq n$ , without loss of generality. We have shown in the text that

$$MI(X;Y) = H(Y) - H(Y|X). \quad (A-4)$$

Since both MI and entropy  $H$  are always positive, the maximum value of MI occurs when  $H(Y|X)$  is zero.

The conditional entropy is defined by  $H(Y|X) \equiv -\sum_{x,y} P(x,y) \log_2 P(y|x)$ . Since  $P(x,y) = P(y|x)P(x)$ , the conditional entropy vanishes when, for all possible values, either  $P(y|x) = \{0,1\}$  or  $P(x) = 0$ . The second condition means that some value of  $x$  never occurs; let us remove it from the set of values. The first condition implies that for a given  $x$ , the condition  $Y=y_j$  either always, or never, occurs. In this case we can relabel the possible values of  $x$  and  $y$  as  $x \in \{x_k^j\}$  and  $y \in \{y_k\}$  with each  $y_k$  always being chosen when a member of the corresponding subset of  $x_k^j$ 's is chosen.

For example, consider the problem where  $x \in \{1, 2, 3, 4, 5, 6\}$ ,  $y \in \{green, gold\}$ , and whenever  $x$  is odd (even),  $y$  is always *green* (*gold*), respectively. In this problem, we can relabel the values  $x \in \{1, 2, 3, 4, \dots\}$  to  $\{x_{green}^1, x_{gold}^1, x_{green}^2, x_{gold}^2, \dots\}$ . Then  $P(Y=green|X=x_{green}^k)=1$ , and  $P(Y=green|x_{gold}^k)=0$ , etc., for all possible  $k$ .

In the general case, we have  $P(y_i|x_k^j) = \delta_{ik}$ . This implies

$$P(x_k^j, y_i) = P(y_i|x_k^j)P(x_k^j) = P(x_k^j)\delta_{ik}. \quad (\text{A-5})$$

Using the definition for conditional entropy,

$$H(Y|X) = - \sum_{i,j,k} P(x_k^j)\delta_{ik} \log_2 \delta_{ik} = \sum_{j,k} P(x_k^j) \log_2 1 = 0 \quad (\text{A-6})$$

Then  $MI(X;Y)=H(Y)$ . As was shown previously, the maximum value of MI occurs when all values of  $y$  are equally probable ( $P(y)=1/m$  where  $m$  is the number of possible values of  $y$ ); in that case, using the definition of  $H(Y)$ ,

$$MI(X;Y) = - \sum_i P(y_i) \log_2 P(y_i) = - \sum_{i=1}^m \frac{1}{m} \log_2 \frac{1}{m} = \log_2 m. \quad (\text{A-7})$$

### Naïve Bayesian Classifier Instability

To further illustrate the problems inherent in assuming independence in the Naïve Bayesian Classifier, consider a system of three binary variables, A, B, and C, where C is the class we wish to examine. Following the discussion in the main document, we seek

$$P(C|AB) = \frac{P(AB|C)P(C)}{P(AB)} = \frac{P(A|C)P(B|C)P(C)}{P(A)P(B)} \quad (\text{A-8})$$

where the first step uses Bayes' Theorem and the last step relies on the independence of A and B. For this example, we use the prior  $P(C) = 1/2$  and, for the eight possible combinations of the values A, B and C from  $\{0,1\}$ , use

$$P(AB|C = 0) = \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{bmatrix}, \quad \text{and} \quad P(AB|C = 1) = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}, \quad (\text{A-9})$$

where a the top row represents  $A=0$ , the bottom row  $A=1$ , the left column  $B=0$ , and the right column  $B=1$ .

First, examine the values  $P(C|AB)$  exactly (without the independence assumption). Marginalizing the values given in (A-9) over  $C^{29}$  shows that  $P(AB)=1/4$  for all four possible combinations of  $A$  and  $B$ . Putting all these values into equation (A-8) gives the true classification equations

$$P(C = 0|AB) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \text{and} \quad P(C = 1|AB) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \quad (\text{A-10})$$

This shows, for example, that when  $A=0$  and  $B=0$  (the top left entry),  $C$  cannot be 0 and is certain to be 1—perfect classification.

If we use the independence assumption, a different result arises. If we calculate  $P(A|C)$  and  $P(B|C)$  by marginalizing (A-9) over the unneeded variable, we find

$$P(A|C = 0) = \sum_{B=0,1} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 2 \end{bmatrix}, \quad (\text{A-11})$$

and the same for  $P(B|C)$ . Therefore, both of these terms equal  $1/2$  for all values of  $A$ ,  $B$ , and  $C$ . Note also that  $P(A) = P(B) = 1/2$

If we then apply these values to the right hand side of (A-8), after the independence assumption, we find a classification result

$$P(C|AB) = \frac{P(A|C)P(B|C)P(C)}{P(A)P(B)} = \frac{\frac{1}{2} \frac{1}{2} \frac{1}{2}}{\frac{1}{2} \frac{1}{2}} = \frac{1}{2} \quad (\text{A-12})$$

---

<sup>29</sup> Marginalize over  $C$  by summing the two matrices and normalizing to 1.

for *all* values of A, B, and C. This represents a complete *inability* to classify C given A and B. Thus, the independence assumption has rendered the NBC completely unusable for this set of values.



## APPENDIX B: MATLAB CODE

The native MATLAB code used to produce the results throughout this text is presented below. All subroutines that are not native MATLAB functions are included, however, some calls to MATLAB functions require the Statistics Toolbox.

We have maintained the original MATLAB code coloration and number of characters per line to increase readability.

### Naïve Bayesian Classifier Code

The MATLAB code in this section is a standalone function (intended to be repeated a number of times) that takes two data sets, one for each of two choices of class (such as disease or normal) and attempts to find a feature set. Details are in the code itself or in Chapter 5: Application of the Naïve Bayesian Classifier.

#### Contents

- Options Section
- Set up Section
- Normalization
- Initial Classification Section
- Start Removing Peaks
- Start Adding Peaks
- Combine Groups
- Remove Outliers
- Count number correct
- Perform the Cross Validation
- Perform the Naive Bayesian Classification
- Create n groups for Cross Validation
- Remove Zeros left by array split up
- Split up groups for cross validation
- Timer Tool

```

function [FinalError PeakList ANorm BNorm] = NBC (ClassA, ClassB, xValReps,..
    OutlierRemoval, RemovePeaks, PeaksToLeave, Normalize)
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% NBC (a wrapper) finds a feature set through forward or backward selection
%
% DESCRIPTION
%     NBC implements a Naive Bayesian Classifier to select features.
%     Given two groups of cases from two different classes, it sets up a
%     cross validation scheme and then begins to remove (or add)
%     features, looking at the ability of each feature's removal to
%     decrease error to determine if that removal should be permanent.
%     You then have the choice of removing highly misclassified samples
%     (if OutlierRemoval~=0). The integer flag RemovePeaks controls
%     whether features are removed one at a time to leave the (integer)
%     PeaksToLeave most important ones (RemovePeaks=1). Alternately, if
%     RemovePeaks=-1, features are added. Setting the flag to zero skips
%     this step.
%
%     The function returns the list of peaks that were left after feature
%     removal (or found during feature addition) as well as the
%     corresponding error rates at each step. These error rates are from
%     nfold cross validation, repeated xValReps (integer) times. Other
%     options are listed immediately below.
%
% USAGE
%     [Error PeakList]=MainProgram(A,B, reps,outlier,remove,leave,norm);
%
% INPUTS
%     A, B: Data is stored in two arrays of continuous data, separated by
%           class, with the cases arranged in the rows and the features, or
%           variables, in columns. A is typically the normal group and B
%           the disease group. The (i,j) value is "intensity of Peak j for
%           Case i."
%     reps: times to repeat the whole process
%     outlier: true if outlier removal is desired, otherwise false.
%     remove: 1 for backward selection, -1 for forward selection
%     leave: number of final features desired
%     norm: Logical, whether to perform total ion normalization
%
% INTERNAL SETTINGS
%     Number of Bins: for discretization, number of bins & build
%                     probability distribution. Choices are 2,4, or 6 for discrete; 0
%                     for continuous (no bins). 2 works best for highly separated
%                     features, 6 or 0 for overlapping groups.
%     n: For n-fold cross-validation
%     Threshold = The probability threshold for declaring the class, e.g.
%                 0.5 means "if P(Class)>.5, Class=disease"
%
% OUTPUTS
%     FinalError: List of the error rate after each removal/addition
%     PeakList: List of which of the "leave" peak remained
%     ANorm, BNorm: Data with total ion normalization applied
%
% CALLED FUNCTIONS
%     CombineGroups: Creates a population from the two inputs
%     CrossValidate: Divides the data into test and training groups
%     PerformBayes: Performs the actual NBC analysis to find P(C|Data)
%     CountCorrect: Finds which cases were classified correctly
%     RemoveOutliers: If desired, removes cases that appear as outliers

```

## Options Section

```
% See explanation at top
NumberOfBins = 6;
n = 10;
Threshold = .5;
```

## Set up Section

```
start = clock;
% Check the sizes of the arrays
[NumA ColsA] = size(ClassA);
[NumB ColsB] = size(ClassB);
if ColsA ~= ColsB
    Error = 'Input Arrays must have same width' %#ok<NOPRT>
    disp(Error);
else
    Cols = ColsA;
    clear ColsA ColsB
end
% Set a bound on feature removal
PeaksToRemove = max ([0 Cols-max([2 PeaksToLeave])] );
Population = CombineGroups(ClassA,ClassB); % see function at end
```

## Normalize

```
%Normalize the sum of each row of data to to the average of the population
if Normalize
    TotIonCt=mean(sum(Population,2)); %Population mean total per row
    factA=sum(ClassA,2)/TotIonCt;
    factB=sum(ClassB,2)/TotIonCt;
    for i=1:NumA
        ClassA(i,:)=ClassA(i,+)/factA(i);
    end
    for i=1:NumB
        ClassB(i,:)=ClassB(i,+)/factB(i);
    end
end
ANorm=ClassA;
BNorm=ClassB;
```

## Initial Classification Section

```
%Find and display Population Classification
Population = CombineGroups(ClassA,ClassB); % see function at end
PopProbInClassB = PerformBayes (Population, ClassA,ClassB, NumberOfBins);
PopCorrectness = PopProbInClassB; clear PopProbInClassB;
PopCorrectness (1:NumA) = 1 - PopCorrectness (1:NumA);
NominalErrorRate = CountCorrect (PopCorrectness,Threshold);
figure();
hist(PopCorrectness*100,100); figure(gcf); axis ([0 50 0 NumA/2]);
title({'Worst Classified Patients';'All Tokens';...
    ['Nominal Error Rate =',num2str(NominalErrorRate*100),'%']});
xlabel ('Percent Correct');
ylabel ('Number of Patients');
```

```
% Optionally remove cases that are badly misclassified
if OutlierRemoval
    prompt = {'Threshold (in percent) for removing misclassified patients'};
    dlg_title = 'Remove Outliers';
    num_lines = 1;
    default_answer = {'0'};
    OutlierRemovalThreshold = str2num(cell2mat..
        (inputdlg(prompt,dlg_title,num_lines,default_answer)))/100;
    clear prompt dlg_title num_lines default_answer;
```

```

NumOut=NumA+NumB;          %Placeholder for number removed
[ClassA ClassB NumA NumB] = RemoveOutliers (ClassA, ClassB, NumA,...
    NumB, PopCorrectness, OutlierRemovalThreshold);
axis ([0 100 0 NumA+NumB/2]);
title({'Original Patients, All Tokens, ',num2str(NumberOfBins),...
    ' Bins'];['Nominal Error Rate =',...
    num2str(NominalErrorRate*100),'%']);

% Redo Nominal Error Rate
Population = CombineGroups(ClassA,ClassB);%see function at end
PopProbInClassB =...
    PerformBayes (Population, ClassA, ClassB, NumberOfBins);
PopCorrectness = PopProbInClassB; clearPopProbInClassB;
PopCorrectness (1:NumA) = 1 - PopCorrectness (1:NumA);
NominalErrorRate = CountCorrect(PopCorrectness,Threshold);
xValErrorRate =...
    CrossValidate (ClassA, ClassB, n, NumberOfBins, xValReps);

NumOut=NumOut-(NumA+NumB);
figure(); hist(PopCorrectness*100,100); axis ([0 100 0NumA+NumB]);
title({'num2str(NumOut),' Patients Removed, All Tokens, ',...
    num2str(NumberOfBins),' Bins'];...
    ['Nominal Error Rate =',num2str(NominalErrorRate*100),'%'];...
    ['Cross Validated Error Rate =',num2str(xValErrorRate*100),'%']});
xlabel ('Percent Correct');
ylabel ('Number of Patients');
msgbox('Outliers removed. Details saved in OutliersRemoved.mat');
end % of optional outlier removal

Start Removing Peaks
if RemovePeaks == 1
    % Remove Peaks one at a time, report error rates
    PeakNumber = 1:Cols;

    % Build a data set without that peak and find the error
    for RemovePeak = 1:PeaksToRemove
        NumCols = Cols+1-RemovePeak;
        PeakxValErrorRate = zeros(1,NumCols);
        for Peak = 1:NumCols
            ClassAMinus = ClassA;
            ClassAMinus (:,Peak) = [];
            ClassBMinus = ClassB;
            ClassBMinus (:,Peak) = [];
            [SampleCorrectness(Peak,:) PeakxValErrorRate(Peak)]..
                = CrossValidate (ClassAMinus,...
                    ClassBMinus, n, NumberOfBins, xValReps);%#ok<*AGROW>
        end

        % Try to capture peak discrimination
        CurrentPeakError = zeros (1,Cols);
        for p = 1:Peak
            CurrentPeakError (PeakNumber (p))=PeakxValErrorRate (p);
        end
        [LowErrorRate BestPeakToRemove] = min(PeakxValErrorRate);
        PeakRemoved = PeakNumber (BestPeakToRemove);
        ClassA(:,BestPeakToRemove) = [];
        ClassB(:,BestPeakToRemove) = [];
        PeakNumber (BestPeakToRemove)=[];
        PeaksOut (RemovePeak,1) = RemovePeak;
        PeaksOut (RemovePeak,2) = PeakRemoved;
    end
end

```

```

        PeaksOut(RemovePeak,3) = LowErrorRate;
        DisplayElapsedTime (PeakRemoved, LowErrorRate, PeaksToRemove,..
            RemovePeak,start);
        FinalError(PeaksToRemove)=LowErrorRate;
    end
    PeakList=PeakNumber;
end
Start Adding Peaks
if RemovePeaks == -1 % Add Peaks one at a time, report error rates
    PeakNumber = 1:Cols;
    ClassAFinal=zeros(NumA,PeaksToLeave);
    ClassBFinal=zeros(NumB,PeaksToLeave);
    for AddPeak = 1:PeaksToLeave % Repeat until desired number of peaks
        NumCols = Cols+1-AddPeak; % Initialize number of peaks to test
        PeakxValErrorRate = zeros(1,NumCols);% Initialize current errors
        ClassAPlus=ClassAFinal(:,1:AddPeak);
        ClassBPlus=ClassBFinal(:,1:AddPeak);
        for Peak = 1:NumCols
            ClassAPlus(:,AddPeak) = ClassA(:,Peak);
            ClassBPlus(:,AddPeak) = ClassB(:,Peak);
            [SampleCorrectness(Peak,:) PeakxValErrorRate(Peak)]...
                = CrossValidate (ClassAPlus,...
                    ClassBPlus, n, NumberOfBins, xValReps);
        end
        CurrentPeakError = zeros (1,Cols);
        for p = 1:Peak
            CurrentPeakError (PeakNumber (p))=PeakxValErrorRate (p);
        end
        [LowErrorRate BestPeakToAdd] = min(PeakxValErrorRate);
        ClassAFinal(:,AddPeak)=ClassA(:,BestPeakToAdd);
        ClassBFinal(:,AddPeak)=ClassB(:,BestPeakToAdd);
        PeakAdded = PeakNumber(BestPeakToAdd);
        ClassA(:,BestPeakToAdd) = [];
        ClassB(:,BestPeakToAdd) = [];
        PeakNumber(BestPeakToAdd)=[];
        PeaksAdded(PeakAdded,1) = AddPeak;% Order Added
        PeaksAdded(PeakAdded,2) = PeakAdded;% Specific Peak added
        PeaksAdded(PeakAdded,3) = LowErrorRate;% Error Value that Peak
        DisplayElapsedTime (PeakAdded, LowErrorRate, AddPeak,..
            PeaksToLeave,start);
        PeakList (AddPeak)=PeakAdded;
        FinalError(AddPeak)=LowErrorRate;
    end
end
end %function
Combine Groups
function population = CombineGroups (clsA, clsB)
% Combines two groups together to make a population
rowsA = size (clsA,1);
rowsB = size (clsB,1);
population (1 : rowsA, :) = clsA;
population (rowsA+1 : rowsA+rowsB, :) = clsB;
end
Remove Outliers
function [A B nA nB]...
    = RemoveOutliers(clsA, clsB, numA, numB, correct, threshold)
% Removes rows in clsA and clsB whose corresponding correctness is below
% the threshold.
nA=numA;

```

```

nB=numB;
correctness = correct;
removed = cell (numA+numB,1);
for i = numB:-1:1
    if correct(i+numA)<threshold
        clsB(i,:)=[];
        nB=nB-1;
        correctness(i+numA)=[];
        removed(numA+i,1) = {'removed because correctness was ',...
            num2str(100*correct(i+numA)),'%'};
    else
        removed(numA+i,1) = {'not removed'};
    end
end

for i = numA:-1:1
    if correct(i)<threshold
        clsA(i,:)=[];
        nA=nA-1;
        correctness(i)=[];
        removed(i,1) = {'removed because correctness was ',...
            num2str(100*correct(i)),'%'};
    else
        removed(i,1) = {'not removed'};
    end
end

A=clsA;
B=clsB;
save 'OutliersRemoved' removed;
end

```

### Count number correct

```

function errorrate = CountCorrect(correctvector,threshold)
% Counts the number of entries in correctvector that are above threshold
len=max(size(correctvector));
count=sum(correctvector>threshold);
errorrate=1-(count/len);
end

```

### Perform the Cross Validation

```

function [correctnesstable xValErrorRate] = CrossValidate (clsA, clsB,..
    n, NumberOfBins, reps)
% This function manages the overall cross validation, splitting the data
% into n groups and then choosing one group at a time to be the test group.
% The Bayes analysis is done inside the cross validation attempt.

for r = 1:reps
    % Split each class into n subgroups for nfold cross validation

    [nGroupsA RowsInGroupsA]=nfold(n, clsA);% This function appears below
    [nGroupsB RowsInGroupsB]=nfold(n, clsB);% This function appears below

    % Now we iterate the n groups, choosing n1 groups to "learn" from and
    % the other group to classify.
    position = 0;
    for i = 1 : n
        NumClassA = RowsInGroupsA (i);
        NumClassB = RowsInGroupsB (i);

        % Combine n-1 of the groups to train on, the other to test
    end
end

```

```

% This function appears below
[Test TrainA TrainB] = CreateValGroups(i, n, nGroupsA, nGroupsB);

% Send these groups to the Bayesian Classifier
PrBP = PerformBayes(Test, TrainA, TrainB, NumberOfBins);

% Build the correctness table
correctnesstable(position+1:position+NumClassA)=..
    1-PrBP(1:NumClassA);
position = position+NumClassA;
correctnesstable(position+1:position+NumClassB)=..
    PrBP(NumClassA+1:NumClassA+NumClassB);
position = position + NumClassB;
end

% Call a function that returns an error rate for the correctness table
TrialErrorRate(r) = CountCorrect(correctnesstable, .5); %#ok<AGROW>
end
xValErrorRate = mean(TrialErrorRate);
end

```

### Perform the Naive Bayesian Classification

```

function PrBP = PerformBayes (testgroup, groupa, groupb, NumBins)
% This function reads in a pair of training groups, creates a probability
% distribution table, and then uses that to classify a test group. The
% array that is returned has the probability (frm 0 to 1) that the
% corresponding element in the test group is in class B

% First determine number of data sets (rows) and elements per data set
% (cols) for each class
[RowsA Cols] = size(groupa);
RowsB = size(groupb,1);
Rows = RowsA + RowsB;
Pa = RowsA/Rows;
Pb = RowsB/Rows;
Prior = Pb;
[RowsT ColsT] = size(testgroup);
PopBins = zeros (NumBins+1,Cols);
ProbDistA = zeros (NumBins+1,Cols);
ProbDistB = zeros (NumBins+1,Cols);
PopProbDist = zeros (NumBins+1,Cols);

% Now Combine the classes into a population
Population(1:RowsA, :)=groupa;
Population (RowsA + 1 : RowsA + RowsB, :) = groupb;
PrPeaksA = zeros (RowsT, Cols);
PrPeaksB = zeros (RowsT, Cols);

% Create Row Vectors with mean and standard dev for each peak
AvgColValue=mean(Population);
StandardDev=std(Population);

if NumBins ==0 % Do Continuous Case
    MeanA=mean(groupa);
    MeanB=mean(groupb);
    StDevA=std(groupa);
    StDevB=std(groupb);
    % Calculate the probability (from a normal distribution) of getting
    % that value. Use 1% as a minimum.
    for c = 1:Cols

```

```

    for r = 1:RowsT
        PrPeaksA(r,c)=...
            max([exp(-(testgroup(r,c)-MeanA(c))^2/StDevA(c)^2) .01]);
        PrPeaksB(r,c)=...
            max([exp(-(testgroup(r,c)-MeanB(c))^2/StDevB(c)^2) .01]);
    end
end
else % Do Discrete case
    for i = 1 : Cols
        % Create bins - either 2, 4, or 6 (see below)
        if NumBins == 2 % Create Bins for above and below mean
            Bins=[-inf, AvgColValue(i),inf];
        elseif NumBins == 4 % Create additional bins 1 std dev above/below
            Bins=[-inf,AvgColValue(i)-StandardDev(i),AvgColValue(i),...
                AvgColValue(i)+StandardDev(i), inf];
        elseif NumBins ==6 %Create bins +/- .5,1.2 std devs from mean
            Bins=[-inf, AvgColValue(i)-1.2*StandardDev(i),...
                AvgColValue(i)-.5*StandardDev(i),...
                AvgColValue(i), AvgColValue(i)+.5*StandardDev(i),...
                AvgColValue(i)+1.2*StandardDev(i),inf];
        else
            disp('Number of Bins must be 2, 4, or 6');
        end
        PopBins(:,i)=Bins;

        % Bin each peak in each set into the bins created above store the
        % bins for each peak in PopBins, store the normed histogram in
        % PopProbDist
        ClassProbDistA=histc(groupa(:,i),Bins);
        ClassProbDistB=histc(groupb(:,i),Bins);
        ProbDistA(:,i)=ClassProbDistA/RowsA;
        ProbDistB(:,i)=ClassProbDistB/RowsB;
    end % creating Bins and population distributions

    % Find the Probability distributions
    % ProbDistA and B are NumBins x "number of peaks" arrays. The
    % probability distribution lookup table for each peak's set of bins is
    % in each column. This is the likelihood for one peak "i"
    % Pr(Pi|Class). BinnedData has the bh that each patient's peak's fall
    % within (1-6). Prior is a scalar (0-1) that is the Pr (B) given only
    % population info.

    % Shave off bottom row which is all zeros
    LastRow = size(PopProbDist,1);
    ProbDistA(LastRow,:)=[];
    ProbDistB(LastRow,:)=[];

    % Bin the test data
    BinnedData = zeros (RowsT, ColsT);
    for i = 1 : RowsT
        for j = 1 : ColsT
            if testgroup (i,j) < PopBins (2,j)
                BinnedData (i,j) = 1;
            elseif testgroup (i,j) < PopBins (3,j)
                BinnedData (i,j) = 2;
            elseif testgroup (i,j) < PopBins (4,j)
                BinnedData (i,j) = 3;
            elseif testgroup (i,j) < PopBins (5,j)
                BinnedData (i,j) = 4;
            elseif testgroup (i,j) < PopBins (6,j)

```



```

        BinnedData (i,j) = 5;
    else
        BinnedData (i,j) = 6;
    end
end
end
end
end

% Now examine the test data
for i = 1 : RowsT
    for j = 1 : ColsT
        % Build two arrays with each entry being the Pr(that bin|Class)
        % for all peaks for all patients. The pointer on where to look is
        % the array BinnedData (i,j) -- a number 1 to 6 representing the
        % bin for that peak. It becomes the row that is looked up in the
        % probability distribution table. Set a min of 1% to prevent zero
        % probability.
        PrPeaksA (i,j) = max([ProbDistA(BinnedData(i,j),j) .01]);
        PrPeaksB (i,j) = max([ProbDistB(BinnedData(i,j),j) .01]);
    end
end
end %choice of discrete or continuos

% Now find the Pr (P|Class) by taking the product Pr(Pi|Class). The
% product ends up as a column vector with each row representing the
% Pr(P|Class) for that patient. This is the likelihood.
PrPA = prod (PrPeaksA,2);
PrPB = prod (PrPeaksB,2);

% Compute evidence - Prob (Peaks) marginalizing across groups
PrP = (PrPA*Pa) + (Pb*PrPB);

% Use Bayes' rule to calculate the posterior Prob (class|Peaks). This
% Posterior is returned as the output of the function. Only the
% Prob of being in class B is returned. Pr(A|P) is that subtracted from 1.
PrBP = zeros (RowsT,1);
for i = 1 : RowsT
    if PrP(i)==0
        PrBP (i) = 0;
    else
        PrBP(i) = (PrPB(i) * Prior)/PrP(i);
    end
end
end
end

```

### Create n groups for Cross Validation

```

function [Test TrainA TrainB] =...
    CreateValGroups(i,n,ClassAGroups,ClassBGroups)
% Given two sets of groups n x Row x Col, selects the ith group as a test
% group and removes that group from the set.

% Create three 1 x C arrays
Cols = size(ClassAGroups,3);
Test = zeros(1,Cols);
TrainA = Test;
TrainB = Test;
Coltest=Cols;

% Create the Test Group from the ith group of the two input arrays
TestA (:,:) = ClassAGroups(i,:,:);
TestB (:,:) = ClassBGroups(i,:,:);

```

```

if Cols == 1
    TestA=TestA';
    TestB=TestB';
end
Test = cat (1, TestA, TestB);

% Remove the Test Group from the mix
ClassAGroups(i, :, :) = [];
ClassBGroups(i, :, :) = [];

%Create the two training Groups from the remainder of the input arrays
for j = 1 : n-1
    GroupA (:, :) = ClassAGroups (j, :, :);
    GroupB (:, :) = ClassBGroups (j, :, :);
    if Coltest == 1 %This section makes sure MATLAB handles a single
        % column as a column not a row vector.
        GroupA=GroupA';
        GroupB=GroupB';
        Coltest=0;
    end
    TrainA = cat (1, TrainA, GroupA);
    TrainB = cat (1, TrainB, GroupB);
end

% Because of the uneven size of the arrays, they will have lines of all
% zero which need to be deleted using a function RemoveZeros
Test = RemoveZeros (Test); % This function appears below
TrainA = RemoveZeros (TrainA);
TrainB = RemoveZeros (TrainB);
end

Remove Zeros left by array split up
function ArrayOut = RemoveZeros (ArrayIn)
% Removes any line in ArrayIn that is all zeros* (from bad indexing)
% *careful it only checks if the sum is zero
% Build a vector Hash with the sums of each row.
Rows = size(ArrayIn,1);
Hash = sum(ArrayIn,2);

% Look through Rows and delete any with a sum of zero
for i = Rows:-1:1
    if Hash(i) == 0
        ArrayIn(i, :)=[];
    end
end
ArrayOut = ArrayIn;
end

Split up groups for cross validation
function [ArraysOut RowsInGroup] = nfold(n, ArrayIn)
% nfold splits array into n arrays of nearly equal length and returns
% it as the 3-D array ArraysOut(1) through (n). The last row of some of the
% arrays is all zeros since the input array may not be split evenly
RowsInGroup=zeros(n,1); % Stores how many are in each group

% Determine size of the array
[Rows Columns] = size(ArrayIn);

% Randomize the order of the rows prior to splitting by attaching a random
% vector (random numbers 0-1), sorting by that vector, then deleting it.
SortVector = rand(Rows,1);

```

```

AppendData = [SortVector ArrayIn];
SortData = sortrows(AppendData);
SortData(:,1) = [];

% Find out how many rows go in each final array. RowsLeft counts down as we
% pull rows out into the Output array.
RowsLeft=Rows;
NumInGroups = int8 ((Rows-mod(Rows,n))/n);
ArraysOut=zeros(n,NumInGroups+1,Columns);
for i = 1:n
    % Determine if the array can be split evenly, if not, add one to each
    % group until the split is even
    if (mod(RowsLeft,NumInGroups) == 0)
        RowsInGroup(i) = NumInGroups;
    else
        RowsInGroup(i) = NumInGroups + 1;
    end

    % Determine how many rows are left. For however many rows are in the
    % current group, pull a row out of the main array into a sub array
    RowsLeft=RowsLeft-RowsInGroup(i);
    for j = 1:RowsInGroup(i)
        for k = 1:Columns
            ArraysOut (i,j,k) = SortData(RowsLeft+j,k);
        end
    end
end
end
end

```

### Timer Tool

```

function DisplayElapsedTime(PeakRemoved, LowErrorRate, PeaksToRemove,...
    RemovePeak, starttime)
% This tool keeps the user informed of the progress
ElapsedTime = clock-starttime;
PeaksLeft=PeaksToRemove-RemovePeak;
% RemainTime=((ElapsedTime(4)*3600)+(ElapsedTime(5)*60)+...
% ElapsedTime(6))*(.75)*PeaksLeft/60;
if ElapsedTime(6)<0
    ElapsedTime(6) = ElapsedTime(6)+60;
    ElapsedTime(5) = ElapsedTime(5)-1;
end
if ElapsedTime(5)<0
    ElapsedTime(5) = ElapsedTime(5)+60;
    ElapsedTime(4) = ElapsedTime(4)-1;
end
Note = ['Removing Peak ', num2str(PeakRemoved), ' with error ',...
    num2str(LowErrorRate*100),'%', ' ,...
    num2str(PeaksLeft),...
    ' left, Elapsed time =', num2str(ElapsedTime(4)),...
    'h,', num2str(ElapsedTime(5)), 'm,',...
    num2str(int8(ElapsedTime(6))), 's.'];
disp(Note);
end

```

## Bayesian Network Algorithm

The MATLAB code in this section performs repeated n-fold cross-validation trials. During each cross-validation, an adjacency matrix is saved to document the feature set; a meta-variable adjacency matrix is saved to document the combining of variables; and the overall error rate is recorded and saved.

The first function listed is the main program. Sub-functions called by that one are listed afterwards. There are several references to functions only available to the MATLAB “statistics toolbox,” which is not part of the base MATLAB library.

Details on this algorithm can be found in Chapter 6: Bayesian Network Algorithm.

### Contents

- Initial Processing of Input Structure
- Build network links
- Attempt to find metavariables
- Bayes classification
- Optimized discretization
- Optimized binning search
- Automated MI thresholding
- Find arcs, build adjacency matrix
- Clear irrelevant arcs
- Attempt to prune relevant arcs
- Find all necessary mutual information values
- Coarse optimized binning,
- Find Bayes network parameters
- Find all variable entropies
- Entropy equation implementation
- Find mutual information of a vector with all columns of an array

```
function OutputStructure = BayesianNetworkClassifier (InputStructure)
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% DoTheMath takes a data set and performs feature selection
%
% DESCRIPTION
%     DoTheMath takes a data array, class vector, and other information
```

```

% and builds and assesses a Bayesian network after selecting features
% from within the data array. It is called from the user interface
% "orca.m."
%
% This is the umbrella script that loops a specified number of times
% (see "repeats" below), each time doing a full nfold cross
% validation and recording the results. All input and output data
% are stored in a single data structure, described below.
%
% USAGE
% OutputDataStructure = DoTheMath (InputStructure)
%
% INPUTS
% InputStructure: Data repository with fields: Intensities: Array of
% intensity values of size #cases x #variables Class: Vector of
% length "#cases", with discrete values identifying
% class of each case (may be integer)
% ID: Patient ID array of length #cases, with one or more cols MZ:
% Vector of length "#variables" holding labels for variables Options:
% Logical 6x1 array. Options are:
% 1. Normalize on population total ion count (sum across rows) 2.
% Remove negative data values by setting them to zero 3. After
% normalizing, before binning, average cases with same ID 4. Find
% the MI threshold by randomization 5. Take log(data) prior to
% binning. Negative values set to 1. 6. Remove Low Signal cases
% NOT DONE: 3 Bin (2 Bin if False)
% n: the "n" in n-fold cross validation repeats: Times to repeat the
% whole process (e.g. recrossvalidate) threshold: Factor by which
% the maximum "random" MI is multiplied to
% find the minimum "significant" MI (double, 1.05.0).
%
% OUTPUTS
% OutputDataStructure: all the fields of InputStructure, plus:
% ErrorRate: Vector containing misclassification rate for each repeat
% KeyFeatures: Index to vector MZ that identifies features selected
%
% CALLED FUNCTIONS
%
% InitialProcessing: Applies the options listed above BuildBayesNet:
% Learns a Bayesian Network from the training data ChooseMetaVars:
% Combines variables that may not be physically
% separate molecules.
% TestCases: Given the BayesNet, tests the "test group" to determine
% the probability of being in each class.
% opt3bin: Discretizes continuous data into 3 bins, optimizing MI
% FindProbTables: Learns the values P(C,V) for each variable
% cvpartition and training are MATLAB Statistics toolbox functions.
%
% Initialize Set up (for now) hard coded values:
drop=0.75; % MI loss percentage threshold for testing independence, see
% clipclassconnections
%
% Initial Processing According to options, remove negative values,
% normalize and/or take logarithm of data, replicate average. Store in
% output data structure.
%
% display('Starting Initial Processing of Data');
OutputStructure = InitialProcessing( InputStructure);
display('Initial processing complete.')
display(' ');

```

```

% Get values out of Data structure to be used later
ff=OutputStructure.threshold;
n=double(OutputStructure.n); % for n-fold cross validation; default is 10
repeats=OutputStructure.repeats; % Number of times to repeat CV, default 30
numtrials=repeats*n;
cverrorrate=zeros(numtrials,1);
errorrate=zeros(repeats,1);
data=OutputStructure.Intensities;
class=OutputStructure.Class;

% Find some sizes and initialize variables
[rows cols]=size(data);
% OutputStructure.varlist=zeros(cols,1);
class_predict=zeros(rows,repeats);
class_prob=zeros(rows,repeats);
trial=0; % counter of how many times we perform Bayes Analysis (n*repeats)

% "Repeat Entire Process" Loop

% Repeat all processes the number of times requested
for r=1:repeats
    display(' ');
    display(['Working on repetition number ', num2str(r),' at ',...
            num2str(toc/60),' mins']);

    % Cross Validation Loop This section selects a training and testing
    % group out of the data by dividing it into n groups, and using #1 of
    % those for training and 1 for testing. MATLAB (ver. 2008a or later)
    % has a built in class for this. See MATLAB documentation for
    % "cvpartition" and "training."
    cvgroups = cvpartition ( class, 'kfold', n );

    for cv = 1:n % for each of n test groups, together spanning all cases
        trial=trial+1; % Keep track of each trial
        display(['      Working on cross-validation number ',num2str(cv),...
                ' of ',num2str(n)])

        % The next line uses a function inside "cvpartition" called
        % "training" that returns a logical vector identifying which cases
        % to use as the training group in cross validation.
        traingrpindex=training(cvgroups,cv);

        % Use the vec to extract tng data and class of the tng cases
        traingrp=data(traingrpindex,:);
        traingrpclass=class(traingrpindex,:);

        % The test cases are cases NOT in the training group
        testgrp=data(~traingrpindex,:);
        testgrpclass= class(~traingrpindex,:);

        % Discretize the groups into hi-med-low by optimizing MI(V,C) for
        % each V (feature) in the training data.

        [leftbndry,rightbndry,traingrpbin, maxMI]=opt3bin(traingrp,..
            traingrpclass); %#ok<NASGU>

        % Build an augmented Naive Bayes Network with the training data
        % The adjacency matrix is a logical with true values meaning "there
        % is an arc from row index to column index." The last row

```

```

% represents the class variable.

adjmat = BuildBayesNet( traingrpbin, traingrpclass, ff, drop );
% Find MetaVariables, rebuild data Depending on the option set,
% reduce the V->V links by removing them, or combining them into a
% single variable. The result is a naive Bayesian network with only
% connections C->V.

meta_option=1; % Hard coded for now
classrow=cols+1;
listvec=1:cols; % just a list of numbers
varlist=unique(listvec(adjmat(classrow,:))); % top level vars

if meta_option==1
    [finaldata metas leftbndry rightbndry] =...
        ChooseMetaVars (traingrp, traingrpclass, adjmat);
end

% Bin up the test group using these final results, combining
% variables per the instructions encoded in the "meta" logical
% matrix.

testdata=zeros(size(testgrp));

if isempty(varlist) % in case no links are found
    disp ('Not finding any links yet...');
    errorrate(trial) = 1;
else % if we do find links
    for var = varlist; % each of the parents of metavariables
        metavar=[var listvec(metas(var,:))]; % concatenate children
        testdata(:,var)=sum(testgrp(:,metavar),2); % sum parent/child
    end

    % Now remove empty rows
    finaltestdata=testdata(:,varlist);

    % And bin the result
    testgrpbin=zeros(size(finaltestdata)); %will be stored here
    % Build boundary arrays to test against
    testcases=size(testgrp,1);
    lb= repmat(leftbndry,testcases,1);
    rb= repmat(rightbndry,testcases,1);
    % test each value and record the bin
    testgrpbin(finaltestdata<lb)=1;
    testgrpbin(finaltestdata>=lb)=2;
    testgrpbin(finaltestdata>rb)=3;

    % Populate Bayesian Network

    % With the final set of data and the adjacency matrix, build
    % the probability tables and test each of the test group cases,
    % to see if we can determine the class.

    % Build the probability tables empirically with the training
    % group results
    ptable=FindProbTables(finaldata, traingrpclass);
    prior=histc(class,unique(traingrpclass))/max(size(traingrpclass));

    % find out the probability of each cases being in class 1,2,etc.
    % Cases are in rows, class in columns.

```

```

classprohtable = TestCases (ptable, prior, testgrpbin);
[P_C predclass]=max(classprohtable, [], 2);
class_prob(~traingrpindex,r)=P_C;
class_predict(~traingrpindex,r)=predclass;

%Get the per trial error rate
cverrorrate(trial)= sum(predclass==testgrpclass)/testcases;

%Store some "per trial" data
OutputStructure.Adjacency(trial, :, :)=adjmat;
OutputStructure.MetaVariablesFound(trial, 1:cols, 1:cols)=metas;
ProbTables(trial).TrialTable=ptable; %#ok<AGROW>

end % of finding metavariables

end % of Cross Validation loop

wrong=sum(~(class==class_predict(:,r)));
errorrate(r)=wrong/rows;

end % of repeating entire process loop

% Record the results in the output structure
OutputStructure.ErrorRate=errorrate;% one for each repeat
OutputStructure.CvErrorRate=cverrorrate;% one for each of n*repeats trials
OutputStructure.PredictedClass=class_predict;

% Find out the error for each case
classrep=repmat(class,1,r);
WasIright=classrep==OutputStructure.PredictedClass;
OutputStructure.CasePredictionRate=sum(WasIright, 2)/double(r);

OutputStructure.ClassProbability=class_prob;
OutputStructure.ProbTables=ProbTables;
OutputStructure.SumAdj=squeeze(sum(OutputStructure.Adjacency,1));
OutputStructure.SumMV=squeeze(sum(OutputStructure.MetaVariablesFound,1));
% Save the results as a .mat data file and alert the user.
save results -struct OutputStructure
disp('Results are saved in the current directory as results.mat')

end % of the function

```

### Initial Processing of Input Structure

```

function StructOut = InitialProcessing( StructIn)
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% INITIALPROCESSING Initial Prep of Data from Signal Processing
%
% DESCRIPTION
%     Takes peaklists that have been imported into MATLAB and prepares
%     them for Bayesian Analysis.
%
% USAGE
%     StructOut = InitialProcessing( StructIn)
%
% INPUTS
%     Structure with the following doubletyped arrays Intensities: n x m
%     real-valued array with variables (peaks) in
%     columns, cases (samples) in rows.
%     MZ: List of the labels (m/z value) for each of the variables.

```



```

%           Must be the same size as the number of variables in Intensities
%           Class: Classification of each sample (disease state)- 1 or 2--must
%           be the same size as the number of cases in Intensities ID: Case or
%           patient ID number, same size as class. May have second
%           column, so each row is [ID1 ID2} where ID2 is replicate number.
%           Options (logical): Array of processing options with elements:
%           1. Normalize 2. Clip Data (remove negatives) 3. Replicate
%           Average 4. Auto threshold MI 5. Use Log of Data 6. Remove Low
%           Signal cases NOT DONE: 3 Bin (2 Bin if False)
%
% OUTPUTS
%           DataStructure: MATLAB data structure with the following components:
%           RawData: Intensities as input ClipData: RawData where all
%           values less than 1 are set to 1 NormData: ClipData normalized
%           by total ion count, i.e.
%           divided by the sum of all variables for each case
%           LogData: Natural logarithm of NormData Class, MZ: Same as input
%           ID: Single column. If replicates are not averaged, the entries
%           are now ID1.ID2. If replicates averaged, then just ID1
%           DeltaMZ: difference in peak m/z values to look for adducts
%           RatioMZ: ratios of m/z values to look for satellites
%
% CALLED FUNCTIONS
%           None. (cdfplot is MATLAB "stat" toolbox)
%
% Initialize Data
% find the size, create the output structure, and transfer info

[rows cols]=size (StructIn.Intensities);
StructOut = StructIn;
StructOut.RawData = StructIn.Intensities;

% Option 2: Clip Negatives from data
% set values below 0 to be 1 because negative
% molecule counts are not physically reasonable
% 1 is chosen rather than 0 in case log(data) is used Note: the decision to
% do this before normalization was based on discussions with Dr. William
% Cooke, who created the data set.

if StructOut.Options(2)
    StructOut.Intensities(find(StructOut.Intensities<1))=1; %#ok<FNDSB>
end

% Option 6: Removal of Cases with Low Signal
% find the sum of all values for each row, then normalize each row to
% account for the effects of signal strength over time and other
% instrumental variations in total strength of the signal

% Find the total ion count for each case, then the global average.
% Determine a correction factor for each case (NormFactor)
if StructOut.Options(1) || StructOut.Options(6)
    RowTotalIonCount=sum(StructOut.Intensities, 2);
    AvgTotalIonCount=mean(RowTotalIonCount); %Population average
    NormFactor=AvgTotalIonCount./RowTotalIonCount; %Vector of norm factors
    StructOut.NormFactor=NormFactor; %save this in the structure
end
% If Remove Low Signal is desired, interact with user to determine
% threshold, then remove all cases that are below the threshold.

if StructOut.Options(6)

```

```

figure(999);
cdfplot(NormFactor);
title('Cumulative Distribution of Normalization Factors');

% Request cutoff

text(1.3,0.5,['Click on the graph where you want';...
    'the normalization threshold      ?...
    'Cases with high norm factor (or  ?...
    'low signal) will be removed.    !]);
[NormThreshold, Fraction] = ginput(1);
display([num2str(floor((1-Fraction)*100)), '% of cases removed']);
close(999);
TossMe=find (NormFactor>NormThreshold);%Low signal cases

% Now record, then remove, those cases with low signal

StructOut.LowSignalRemovedCases=StructOut.ID(TossMe,:);
StructOut.LowSignalRemovedCasesNormFactors=NormFactor(TossMe);
StructOut.Intensities(TossMe,:)=[];
StructOut.ID(TossMe,:)=[];
StructOut.Class(TossMe,:)=[];

end

% Option 3: Replicate Average This option causes cases with same ID numbers
% to be averaged, peak by peak.

if StructOut.Options(3) %Replicate Average
    % Collapse to unique IDs only, throw out replicate ID column
    StructOut.Replicate_ID=StructOut.ID;%Save old data
    StructOut.Replicate_Class=StructOut.Class;

    newID=unique(StructOut.ID(:,1));% List of unique IDs
    num=size(newID,1); %how many are there?
    newClass=zeros(num,1); % Holders for extracted class, data
    newData=zeros(num,cols);
    for i=1:num % for each unique ID
        id=newID(i); % work on this one
        cases=find(StructOut.ID(:,1)==id); % Get a list of cases with this ID
        newClass(i)=StructOut.Class(cases(1));% save their class
        casedata=StructOut.Intensities(cases, :);% get their data
        newData(i,:)=mean(casedata, 1);% and save the average
    end
    StructOut.Intensities=newData;
    StructOut.Class=newClass;
    StructOut.ID=newID;
    clear newID newClass newData
else % If replicates exist, combine the 2 column ID into a single ID
    ID= StructOut.ID;
    if min(size(ID))==2
        shortID=ID(:,1)+(ID(:,2)*.001); % Now single entry is ID1.ID2
        StructOut.OldID=StructOut.ID;
        StructOut.ID=shortID;
        clear ID shortID
    end
end

end

% Option 1: Normalize total ion count Apply the normalization factor to

```

```

% each row to normalize total ion count. We'll recalc norm factors in case
% data was replicate averaged.
if StructOut.Options(1)
    RowTotalIonCount=sum(StructOut.Intensities, 2);
    AvgTotalIonCount=mean(RowTotalIonCount);%Population average
    NormFactor=AvgTotalIonCount./RowTotalIonCount;%Vector of norm factors
    StructOut.NormFactor=NormFactor; %save this in the structure
    NFmat=repmat(NormFactor, 1, cols);% match size of Intensities
    StructOut.Intensities=StructOut.Intensities.*NFmat;
    clear NFmat RowTotalIonCount AvgTotalIonCount NormFactor;
end

% Option 5: Work with log (data)

if StructOut.Options(5)
    StructOut.Intensities=log(StructOut.Intensities);
end

% end function

end

Build network links
function adjacency = BuildBayesNet( data, class, ffactor, drop )
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% BuildBayesNet selects features and metafeatures based on mutual info.
%
% DESCRIPTION
%     This function takes a set of training data and an additional
%     variable called "class" and tries to learn a Bayesian Network
%     Structure by examining Mutual Information. The class variable C is
%     assumed to be the ancestor of all other variables V. Arcs from C
%     to V are declared if  $MI(C;V) >> z$ , where  $z$  is a maximum expected MI
%     of similar, but random data...multiplied by a "fudge factor." Arcs
%     from  $V_i$  to  $V_j$  are similarly declared. Then various tests are
%     performed to prune the network structure and combine variables that
%     exhibit high correlations. Finally the network is pruned to be a
%     Naive Bayesian Classifier, with only  $C \rightarrow V$  arcs remaining.
%
% USAGE
%     network_structure = BuildBayesNet( training_data, class )
%
% INPUTS
%     training_data: cases in rows, variables in cols, integer array
%                   containing the data used to build the Bayes net
%     class: the known class variable for each case (1:c col vector)
%     ffactor: multiple of auto MI to use to threshold  $C \rightarrow V$  connections
%     drop:
%
% OUTPUTS
%
%     adjmatrix: a matrix of zeros and ones, where one in row  $i$ , column  $j$ 
%               denotes a directed link in a Bayesian network between
%               variable  $i$  and variable  $j$ . The class variable is the last
%               row/column.
%
% CALLED FUNCTIONS
%
%     automi: finds an MI threshold based on data findmutualinfos: finds

```

```

%      all values MI(V;C), MI(V;V) and MI(V;C|V)

% Initialize

% Initialize the network object and some constants
network.data=data;
network.class=class;

automireps=10; %times to repeat the auto MI thresholding to find avg.

% Check the sizes of various things
[rows cols]=size(data); %#ok<NASGU>
cases=max(size(class));
if rows==cases
    clear cases
else
    disp('# of rows in the data and class must be equal.')
    return
end

% network.adjmat=zeros(cols+1); % all variables plus class as last row/col
dataalphabet=max(size(unique(data))); % number of possible values of data
classalphabet=max(size(unique(class))); % Number of values of class

% Step 0: Find all the necessary mutual information values, thresholds The
% function below finds all values MI(V;C|V) and other combos needed and
% stores them in the network structure.

[ network.mi_vc, network.mi_vv, network.mi_vc_v ]..
= findmutualinfos( data, class );

% Find a threshold MI by examining MI under randomization
network.vcthreshold = automi( data, class, automireps )*ffactor ;
network.vvthreshold = network.vcthreshold *..
    log(dataalphabet)/log(classalphabet);

% Step 1: Find all the possible arcs. Find the variables with high MI with
% the class, i.e. MI(V,C)>>0 and connect a link in the adjacency matrix
% C->V. Also connect variable Vi,Vj if MI(Vi;Vj)>>0

network.adjmat1=getarcs(network.mi_vc,network.vcthreshold,network.mi_vv,..
    network.vvthreshold);

% Step 2: Prune the variable set by clearing irrelevant features If there
% is no path from V to the class, clear all entries V<->Vi (all i)
network.adjmat2 = clearirrarcs( network.adjmat1 );

% Step 3: Cut connections to class Where two variables are connected to
% each other and also to the class, attempt to select one as the child of
% the other and disconnect it from the class. Use MI(Vi;C|Vj)<<MI(Vi;C) as
% a test.

temp = clipclassconnections (network.adjmat2,..
    network.mi_vc,network.mi_vc_v, drop);

% and once again clear features no longer near class and end function
adjacency= clearirrarcs( temp );

end

```

### Attempt to find metavariables

```
function [finaldata metamatrix leftbound rightbound] = ..
    ChooseMetaVars ( data, class, adj)
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% ChooseMetaVars attempts to combine variables into better variables
%
% DESCRIPTION
%     Finds the V-V pairs in the adjacency matrix, and attempts to
%     combine them into a metavariable with a higher mutual information
%     than either variable alone. If it is possible to do this, it
%     returns a new data matrix with the variables combined.
%
% USAGE
%     [finaldata metamatrix leftbound rightbound] =
%         ChooseMetaVars ( data, class, adj)
%
% INPUTS
%     data: double array of discrete integer (1:n) values, cases in rows
%           and variables in columns.
%     class: double column vector, also 1:n. Classification of each case.
%     adj: Adjacency matrix, #variables+1 by #variables. Last row is
%           class node. Logical meaning "there is an arc from i to j."
%
% OUTPUTS
%     metamatrix: logical whose (i,j) means "variable j was combined into
%               variable i (and erased)"
%     finaldata: The data matrix with the variable combined and rebinned
%     leftbound: The new left boundary (vector) for binning. rightbound:
%               The new right boundary (vector) for binning.
%
% CALLED FUNCTIONS
%     opt3bin: rebins combined variables to determine highest MI.
%
% Intialize
[rows cols]=size(data);
[classrow numvars]=size(adj);
bindata=zeros(rows,cols);
metamatrix=false(cols);

% Create a list of all the variables V to check by examining the adjacency
% matrix's last row, i.e. those with C->V connections
listvec=1:numvars;
varstocheck=unique(listvec(adj(classrow,:)));
l=zeros(1,numvars);
r=zeros(1,numvars);

% Now go through that list, testing each V>W connection to see if adding V
% and W creates a new variable Z that has a higher MI with the class than V
% alone. V is the list above, W is the list of variables connected to a V.

for v=varstocheck % Pull out the W variables connected to V and test
    wlist=unique(listvec(adj(v,:)));
    [l(v), r(v), binned, mitobeat] = opt3bin(data(:,v), class);
    bindata(:,v)=binned;
    if ~isempty(wlist)
        for w=wlist
            newdata=data(:,v)+data(:,w);
            [left, right, binned, newmi] = opt3bin(newdata, class);
```

```

        if newmi>mitobeat
            mitobeat=newmi;
            data(:,v)=data(:,v)+data(:,w);
            metamatrix(v,w)=true; % record the combination
            bindata(:,v)=binned;
            l(v)=left;
            r(v)=right;
        end
    end
end
end
end

```

```

%pull out just the V->C columns from the data matrix.
finaldata=bindata(:,adj(classrow,:));
leftbound=l(adj(classrow,:));
rightbound=r(adj(classrow,:));
end

```

### Bayes classification

```

function classprobs = TestCases( p, prior, data)
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% classprobs uses Bayes rule to classify a case
%
% DESCRIPTION
%     Tests each of a set of data vectors by looking up P(data|class) in
%     a probability table, then finding P(case|class) by multiplying each
%     of those values in a product. Then uses Bayes' rule to calculate
%     P(class|data) for each possible value of class. Reports this as an
%     array of class probabilities for each case.
%
% USAGE
%     classprobs = TestCases( p, prior,data)
%
% INPUTS
%     data: double array of discrete integer (1:n) values, cases in rows
%           and variables in columns.
%     p: 3-D double array of probabilities (c,d,v). The first dimension
%         is the class, the second is the data value, the third is the
%         variable number. The entry is P(var v=value d | class=value c).
%
% OUTPUTS
%     classprobs: 2-D double array whose value is P(class=c|data) for
%                each case. Cases are in rows, class in cols.
%
% CALLED FUNCTIONS
%
%     None.
%
% Intialize
%
% Find the sizes of the inputs and the number of possible values
[cases numvars]=size(data);
classvals=size(p,1);
pvec=zeros(classvals,numvars);
classprobs = zeros(cases, classvals); % holds the classification results
% Find the probabilities

```

```

% Create pvec, an array whose first row is P(V=v|c=1) for each V
for casenum=1:cases
    casedata=data(casenum,:); % The case to be checked
    for c=1:classvals
        for v=1:numvars
            pvec(c,v)=max(p(c,casedata(v),v),.01); % Don't want any zeros
        end
    end
    % Now find P(case|class) for each class by multiplying each individual
    % P(V|C) together, assuming they are independant.

    Pdc=prod(pvec,2);

    % Use Bayes' Rule

    classprobs(casenum,:)=(Pdc.*prior)/sum(Pdc.*prior);

end

end

Optimized discretization
function [l, r, binned, mi] = opt3bin (data, class)
% by Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% FunctionName short description
%
% DESCRIPTION
%     This function takes an array of continuous sample data of size
%     cases (rows) by variables (columns), along with a class vector of
%     integers 1:c, each integer specifying the class. The class vector
%     has the same number of cases as the data. Thefunction outputs the
%     position of the 2 bin boundaries (3 bins) that optimize the mutual
%     information of each variable's data vector with the class vector.
%
% USAGE
%     [l,r,binned, mi]=opt3bin(data,class)
%
% INPUTS
%     data: double array of continuous values, cases in rows and
%           variables in columns. Distribution is unknown.
%     class: double column vector, values 1:c representing classification
%            of each case.
%
% OUTPUTS
%     l      - row vector of left boundary position for each var. r      -
%     row vector of right boundary position for each var. binned data
%     array discretized using boundaries in l and r mi      - row vector of
%     mutual info between each discr. variable
%               and class
%
% CALLED FUNCTIONS
%
%     opt2bin: Similar function that finds a single boundary. This is
%             used as a seed for the 3 bin optimization.
%     looklr: See below.
%
% Intialize
%

```

```

% Variable Prep : find sizes of arrays and create placeholders for locals

steps=150;
[rows cols]=size(data);
boundary=zeros(2,cols);

% Method Find starting point by finding the maximum value of a 2 bin mi.
% Next, go left and right from that position, finding the position of the
% next boundary that maximizes MI.

[mi boundary(1,:)] = opt2bin (data, class, steps, 2);

% We've located a good starting (center) bin boundary. Search L/R for a
% second boundary to do a 3 bin discretization.
[mi boundary(2,:)] = looklr (data, class, boundary(1,:), steps);

% We've now found the optimum SECOND boundary position given the best 2 bin
% center boundary. Now re-search using that SECOND boundary position,
% dropping the original (2 bin). The result should be at, or near, the
% optimal 3 bin position.
[mi boundary(1,:) binned] = looklr (data, class, boundary(2,:), steps);

% from the two boundaries found above, sort the left and right
r=max(boundary);
l=min(boundary);

% Now return the vector of left and right boundaries, the disc. data, and
% max MI found.
end % of function

```

### Optimized binning search

```

function [miout nextboundary binned] = looklr (data, class, startbd, steps)
% given a start position, finds another boundary (to create 3 bins) that
% maximizes MI with the class
[rows cols]=size(data);
farleft=min(data, [],1);
farright=max(data, [],1);
miout=zeros(1,cols);
binned=zeros(rows,cols);
nextboundary=zeros(1,cols);

for peak=1:cols % for each peak/variable separately...

    % discretize this variables' values. Sweep through the possible bin
    % boundaries from the startbd to the furthest value of the data,
    % creating 2 boundaries for 3 bins. Record the binned values in a
    % "cases x steps" array, where "steps" is the granularity of the sweep.
    % The data vector starts off as a column...

    testmat= repmat (data (:,peak), 1, steps); % and is replicated to an array.

    % Create same size array of bin boundaries. Each row is the same.
    checkpointsL= repmat (linspace (farleft (peak), startbd (peak), steps), rows, 1);
    checkpointsR= repmat (linspace (startbd (peak), farright (peak), steps), rows, 1);

    % Create a place to hold the discrete info, starting with all ones. The
    % "left" array will represent data binned holding the center boundary
    % fixed and sweeping out a second boundary to the left; similarly the
    % right boundary starts at "startbd" and sweeps higher.
    binarrayL=ones (rows, steps);

```



```

binarrayR=ones(rows,steps);

% Those in the L test array that are higher than the left boundary-> 2
binarrayL(testmat>checkptsL)=2;
binarrayL(testmat>startbd(peak))=3; % >center boundary -> 3

% Similarly using center and right boundaries
binarrayR(testmat>startbd(peak))=2;
binarrayR(testmat>checkptsR)=3;

% Now at each of those step positions, check MI (var;class).
miout(peak) = 0;

% These vectors hold the MI with each step used to discretize.
miL=MIarray(binarrayL,class) % MI(V;C) using left/center
miR=MIarray(binarrayR,class); % MI(V;C) using center/right

if max(miL)>max(miR) % See which one is the largest
    [miout(peak) index]=max(miL); %record the max mi found
    nextboundary(peak)=checkptsL(1,index); % and record the boundary
    binned(:,peak)=binarrayL(:,index) % and record the discrete data
else
    [miout(peak) index]=max(miR); %record the max mi found
    nextboundary(peak)=checkptsR(1,index); % and record the boundary
    binned(:,peak)=binarrayR(:,index) % and record the discrete data
end

end % of that variable's search. Go to next variable.

end % of the search. Return the best boundary and the associated MI and data

```

**Automated MI thresholding**

```

function threshold = automi( data, class, repeats )
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% automi finds a threshold for randomized MI(V; C)
%
% DESCRIPTION
% Finds the threshold of a data set's mutual information with a class
% vector, above which a variable's MI(class, variable) can be
% expected to be significant. The threshold for mi (significance
% level) is found by taking the data set and randomizing the class
% vector, then calculating MI(C;V) for all the variables. This is
% repeated a number of times. The resulting list of length (#repeats
% * #variables) is sorted, and the 99th percentile max MI is taken
% as the threshold.
%
% USAGE
% threshold = automi( data, class )
%
% INPUTS
% data: double array of discrete integer (1:n) values, cases in rows
% and variables in columns.
% class: double column vector, also:1:n. Classification of each case.
% repeats: the number of times to repeat the randomization
%
% OUTPUTS
% threshold: the significance level for MI(C;V)

```

```

%
% CALLED FUNCTIONS
%
%      MIarray(data,class): returns a vector with MI(Vi;Class) for each V
%      in the data set

% Intialize

% Find the size of the data (cases x variables) and check against class
[rows cols]=size(data);
cases=max(size(class));
if rows==cases
    clear cases
else
    disp('# of rows in the data and class must be equal.')
    return
end

% Repeat a number of times

mifound=zeros(cols,repeats); % stores the results of the randomized MI
for i=1:repeats
    c=class(randperm(rows)); % creates a randomized class vector
    mifound(:,i)=MIarray(data,c); % record MI(Ci;V) in an array
end

% pull off the 99th percentile highest MI
mi_in_a_vector=reshape(mifound,repeats*cols,1); % prctile needs vector
threshold=prctile(mi_in_a_vector,99);

end

Find arcs, build adjacency matrix
function adjacency = getarcs( mvc, vcthreshold, mvv, vvthreshold )
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% GETARCS builds the adjacency matrix for a set of variables
%
% DESCRIPTION
%      By comparing mutual information between two variables to thresholds
%      determined seperately, this function declares there to be an arc in
%      a Bayesian network. Arcs are stored in an adjacency matrix,
%      described below.
%
%      The primary tests are: MI(Vi;Cj)>>vcthreshold : tests for links
%      between Vi and the class MI(Vi;Vj)>>vvthreshold : tests the links
%      between variables
%
% USAGE
%      adjacency = getarcs( mvc, vcthreshold, mvv, vvthreshold )
%
% INPUTS
%      mvc [mvv]: double vector [array] with mutual information between
%      variables and the class [variables and other variables]. The
%      (i,j) entries of mvv are MI(Vi,Vj).
%      vc/vvthreshold: scalar threshold used to test for existence linkz
%
% OUTPUTS
%      adjacency: logical matrix whose entries "1" at (i,j) mean "an arc

```

```

%           exists from the Bayesian network node Vi to Vj." The class
%           variable C is added at row (number of V's + 1). "0" values
%           mean no arc.
%
% CALLED FUNCTIONS
%
%           None.
%
% For more information on the tests and the links, see my dissertation.

% Initialize
numvars=max(size(mvc)); %the number of variables
classrow=numvars+1; %row to store links C->V
adjacency= false(classrow,numvars); %the blank adjacency matrix

% Test for adjacency to class
adjacency ( classrow , : )= mvc > vcthreshold;

% Test for links between variables This test results in a symmetric logical
% matrix since MI (X;Y) is symmetric. To create a directed graph, these arcs
% will need to be pruned.
adjacency ( 1:numvars, 1:numvars ) = mvv > vvhreshold;

end
Clear irrelevant arcs
function adjout = clearirrarcs( adjin )
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% CLEARIRRARCS clears arcs that are not E>V or C->V<->V
%
% DESCRIPTION
%           Given an adjacency matrix with V<->V arcs in a square matrix and an
%           additional row representing E>V (class to variable), this function
%           clears out all V1->V2 arcs where V1 is not a member of the set of
%           V's that are class-connected, i.e. have arcs in the final row.
%
% USAGE
%           adjout = clearirrarcs( adjin )
%
% INPUTS
%           adjin: a logical array where a true value at position (i,j) means
%           that there is an arc in a directed acyclic graph between
%           (variable) i and variable j.
%
% OUTPUTS
%           adjout: copy of adjin with unneeded arcs cleared
%
% CALLED FUNCTIONS
%           None.

% Initialize Find the sizes of the input
[classrow, numvars]=size(adjin);

% Main processing Find out which variables are connected to class
conntocls=(adjin(classrow,:));

% Remove all arcs that don't have at least one variable in this list, e.g.
% all Vi<->Vj such that ~(Vi->C or Vj->C). These are all the entries in the
% adjacency matrix whose i and j are NOT in the list above.

```

```

% Make a matrix with ones where neither variable is in the list above
noconnmat= repmat(~conntocls,numvars,1) & repmat(~conntocls',1,numvars);

% Use that to erase all the irrelevant entries in the square adj matrix, at
% the same time remove the diagonal (arcs Vi<>Vi)
adjout=adjin (1:numvars, 1:numvars)& ~noconnmat & ~eye(numvars);

% Bidirectional arcs are temporarily permitted between nodes connected
% directly to the class, but not between nodes where only one is connected
% to the class- those are assumed to flow G>V1->V2 only. Remove V2->V1.

% Get a matrix of ones in rows that are class connected. V>V arcs are only
% allowed to be in these rows:
parents=repmat(conntocls',1,numvars);
% Remove anything else
adjout=adjout & parents;

% Now add back in the class row at the bottom of the square matrix
adjout(classrow,:)=adjin(classrow,:);

end

Attempt to prune relevant arcs
function adjout = clipclassconnections( adj, mivc_vec,mivcv,droptreshold )
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% clipclassconnections delinks variables from class
%
% DESCRIPTION
%       Where two variables are connected to each other and also to the
%       class, attempt to select one as the child of the other and
%       disconnect it from the class. Use MI(Vi;C|Vj)<<MI(Vi;C) as a test.
%
% USAGE
%       probtable = FindProbTables(data, class)
%
% INPUTS
%       adj: (logical) matrix where "true" entries at (i,j) mean "an arc
%           exists from the Bayesian network node Vi to Vj." The class
%           variable C is added at row (number of V's + 1). "0" values
%           mean no arc.
%       mivc_vec: (double) row vector containing MI(C;Vi) for each variable
%       mivcv: (double) array whose (i,j) entry is MI(Vi,C|Vj).
%       droptreshold: percentage drop from MI(Vj;C) to MI(Vj;C|Vi) before
%           declaring that Vi is between C and Vj.
%
% OUTPUTS
%       adjout: copy of adj with the appropriate arcs removed.
%
% CALLED FUNCTIONS
%       None.

% Intialize
[classrow, numvars]=size(adj);
classconnect=adj(classrow, :); % the last row of adj stores arcs G>V
adjout=false(classrow, numvars); % placeholder for output array

% Identify triply connected arcs

```

```

% First look for pairs that are connected to each other and connected to
% the class.

% Connected to each other: build logical array with (i,j) true if Vi<->Vj
vv_conn=adj(1:numvars, 1:numvars);

% Connected to the class: logical array with (i,j) true if  $\exists$ Vi and C->Vj
vcv_conn=repmat(classconnect, numvars,1) & repmat(classconnect',1,numvars);

% Find all (i,j) with both true
triple_conn = vv_conn & vcv_conn;

% Determine preferred direction on V<->V arcs

% Determine the Vi<->Vj direction by finding the greater of MI(C;i|j) or
% (C;j|i). Greater MI means less effect of the instantiation of i or j.
arcdirection=mivcv > mivcv'; %Only the larger survive
dag_triple_conn=arcdirection & triple_conn;% Wipes out the smaller ->

% find links should NOT be kept under the test above,
linkstoremove=(~arcdirection) & triple_conn;
% and if they are in the connection list, remove them
adjout(1:numvars, 1:numvars)=xor(vv_conn,linkstoremove);

% Now we need to test whether we can remove the link between C and which
% ever V (i or j) is the child of the other. We look for a "significant"
% drop in MI(Vj;C) when instatiating Vi, e.g. MI(Vj;C|Vi)<<MI(Vj;C).
%
% droptreshold of .7, for example, means link breaks if 1st term is less
% than 30% of the second term.
%
% If there is a big drop in MI(C;Vj) when Vi is given, and Vi->Vj exists in
% the DAG, then we can remove the link C->Vj and leave C->Vi->Vj.

% Build an array out of the mivc_vec vector
mivc=repmat(mivc_vec',1,numvars);
% Test for the large drop described above
bigdrop=((mivc-mivcv)./mivc) > droptreshold;
% Test for the big drop and the VV connection
breakconn = bigdrop' & dag_triple_conn;
% If any of the elements in a column of the result are true, remove that
% variable's C->V link, since it is a child.
linkstokeep=~any(breakconn);
adjout(classrow,:)= adj(classrow,:) & linkstokeep;

% With V->V links now only one way, and C->V removed where needed, we can
end
Find all necessary mutual information values
function [ mi_vc, mi_vv, mi_vc_v ] = findmutualinfos( data, class )
% by Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% FINDMUTUALINFOS finds the various mutual info combos among variables.
%
% DESCRIPTION
% Given a set of data (many cases, each with values for many
% variables) and an additional value stored in the vector class, it
% finds various combinations of MI described below in "OUTPUTS."
%
% USAGE

```

```

%      [ mi_vc, mi_vv, mi_vc_v ] = findmutualinfos( data, class )
%
% INPUTS
%      data: A number of cases (in rows), each with a measurement for a
%            group of variables (in columns). The data should be discretized
%            into integers 1 through k. The columns are considered variables
%            V1, V2, ...
%      class: an additional measurement of class C. A column vector of
%            length "cases" with integer values 1,2...
%
% OUTPUTS
%
%      mi_vc: a row vector whose ith value is MI(Vi,C). mi_vv: Symmetric
%      matrix with values MI(Vi,Vj). mi_vc_v: Nonsym matrix with values
%      MI(Vi;C|Vj).
%
% CALLED FUNCTIONS
%
%      findentropies: returns entropy values [e.g. H(Vi,Vj)]

% Intialize

%Find the data size and declare some blank arrays
[rows cols]=size(data);
mi_vv=zeros(cols);
mi_vc_v=zeros(cols);

% Find Entropies and Calculate Mutual Informations

% Find the various entropies needed to calculate the MI's
[ h_c, h_v, h_vc, h_vv, h_vcv ] = findentropies( data, class );

% Calculate the value MI(Vi,C)
mi_vc = h_v + h_c - h_vc;

% For each variable Vj, calculate MI(Vi,Vj) and MI(Vi;C|Vj)

for i=1:cols
    for j=1:cols
        mi_vv(i,j) = h_v(i) + h_v(j) - h_vv(i,j);
        mi_vc_v(i,j) = h_vv(i,j) - h_v(j) + h_vc(j) - h_vcv(i,j);
    end
end

end

end

Coarse optimized binning,
function [mi boundary binneddata] = opt2bin (rawdata, class, steps...
typesearch, minint, maxint)
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% opt2bin finds the best single boundary for each variable to maximize MI
%
% DESCRIPTION
%      This function takes an array of continuous data, with cases in rows
%      and variables in columns, along with a vector "class" which holds
%      the known class of each of the cases, and returns an array
%      "binneddata" that holds the 2 bin discretized data. The
%      discretization bin boundary is found by maximizing the mutual
%      information with the class; the resulting MI and boundary are also

```

```

%         returned. The starting boundaries for the search can be given in
%         the vectors min and max, or either one, or neither, in which case
%         the data values determine the search boundaries.%
%
% USAGE
%     [mi boundary binneddata] = maxMIbin(rawdata, class, typesearch [,
%         min, max])
%
% INPUTS
%     rawdata: double array of continuous values, cases in rows and
%             variables in columns. Distribution is unknown.
%     class: double column vector, values 1:c representing classification
%           of each case.
%     steps: Number of steps to test at while finding maximum MI
%     typesearch =0: starting bndry based on data's actual max/min values
%                 =1: use the value passed in max a maximum (right) value
%                 =-1: use the value passed in min as minimum (left) value
%                 =2: used values passed via max, min
%     the two optional arguments are vectors whose values limit the range
%     of search for each variables boundaries.
%
% OUTPUTS
%
%     mi: row vector holding the maximum values of MI(C;Vi) found
%     boundary: The location used to bin the data to get max MI
%     binneddata: The resulting data binned into "1" (low) or "2" (hi)
%
% CALLED FUNCTIONS
%
%     MIarray: Finds the MI of each col in an array with a separate
%             vector (the class in this case)
%
% Intialize
[rows cols]=size(rawdata);
mi=zeros(1,cols);
boundary=zeros(1,cols);
binneddata=zeros(rows,cols);
currentmi=zeros(steps,cols);

% if not passed, find the left and rightmost possible bin boundaries from
% data

if nargin~=6
    minint=min(rawdata,[],1);
    maxint=max(rawdata,[],1);
elseif typesearch==1
    minint=min(rawdata,[],1);
elseif typesearch==-1
    maxint=max(rawdata,[],1);
elseif typesearch==2
    disp('using passed values')
else
    disp('typesearch must = 0,1,-1,2')
    return
end

% Find best boundary

for peak=1:cols %look at each variable separately

```

```

% Create an array of bin boundary's possible locations min->max
checkpoints= repmat( linspace(minint(peak),maxint(peak),steps),rows,1);

% discretize the variable's values at each of these possible
% boundaries, putting 2's everywhere (value > boundary), 1 elsewhere
binarray=( repmat(rawdata(:,peak), 1, steps)>checkpoints)+1;

% Send this array off to find the MI(C,V) for each possible binning
currentmi(1:steps,peak)=MIarray(binarray,class);

% Now pick out the highest MI, i.e. best bin boundary
[mi(peak) atstep]=max(currentmi(:,peak));
boundary(peak)=checkpoints(1,atstep);

% and record the binned data using that boundary.
binneddata(:,peak)=binarray(:,atstep);
end

end

Find Bayes network parameters
function p=FindProbTables(data, class)
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% FindProbTables estimates the probabilities P(class=c|data=D)
%
% DESCRIPTION
%   Input a training group of data arranged with cases in rows and
%   variables in columns, as well as the class value c for that vector.
%   Each case represents a data vector V. For each possible data value
%   vi, and each variable Vi, it calculates P(C=c|Vi=vi) and stores
%   that result in a 3-D table. The table is arranged with the
%   dimensions (class value, data value, variable number).
%
% USAGE
%   probtable = FindProbTables(data, class)
%
% INPUTS
%   data: double array of discrete integer (1:n) values, cases in rows
%         and variables in columns.
%   class: double column vector, also 1:n. Classification of each case.
%
% OUTPUTS
%   probtable: 3-D array whose (c,d,v) value is P(class=c|data=p) for
%             variable v.
%
% CALLED FUNCTIONS
%
%   None.

% Intialize Find the sizes of the inputs and the number of possible values
[cases numvars]=size(data);
datavals=max(size(unique(data)));
classvals=max(size(unique(class)));
% Build some placeholders and loop indices
p=zeros(classvals, datavals, numvars );% triplet: (class, value, variable#)
databins=1:datavals;
classbins=1:classvals;

```



```

% Find Probabilities For each classification value, extract the data with
% that class
for c=classbins
    datainthatclass=data(class==c,:);% array of just cases with class=c
    % find the percentage of data with each possible data value
    p(c, :, :)=histc(datainthatclass,databins)/cases;
end

end

Find all variable entropies
function [ h_c, h_v, h_vc, h_vv, h_vvc ] = findentropies( data, class )

% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% findentropies finds all the entropies H(V), H(V,V), etc.
%
% DESCRIPTION
% Give a set of data arranged with cases in rows and variables in
% columns, and an additional variable labeled the "class", this
% function returns the entropy of each variable's data, the class'
% data, and the joint entropy of all pairs of two variables, all
% variables with the class, and all pairs of variable and the class.
%
% USAGE
% [ h_c, h_v, h_vc, h_vv, h_vvc ] = findentropies( data, class )
%
% INPUTS
% data: double array of discrete integer (1:n) values, cases in rows
% and variables in columns.
% class: double column vector, also 1:n. Classification of each case.
%
% OUTPUTS
%
% h_v: entropies of the variables, H(Vi), stored in a row vector.
% h_c: scalar entropy of the class vector, H(C) h_vc: vector whose
% ith entry is the joint entropy H(Vi,C) h_vv: matrix whose (i,j)
% entry is the joint entropy H(Vi,Vj) h_vvc: matrix whose (i,j) entry
% is the joint entropy H(Vi,Vj,C)
%
% CALLED FUNCTIONS
%
% entropy (vector, num_poss_vals [vector, numvals,...]) see below

% Initialize
% Find the number of variable (cols) and number of cases, as well as the
% number of possible values (k) and class values l)
[rows cols]=size(data);
k=max(size(unique(data)));% # of possible values of data
l=max(size(unique(class)));% # of possible values of class

% Intialize the output matrices
h_v=zeros(1,cols);
h_vc=zeros(1,cols);
h_vv=zeros(cols,cols);
h_vvc=zeros(cols,cols);

% Main processing Calculate all the various entropy combinations
h_c = entropy (class, l);% see function below

```

```

for i=1:cols
    h_v(i) = entropy (data(:,i), k);
    h_vc(i) = entropy (data(:,i), k, class, 1);

    for j=1:cols
        h_vv(i,j)= entropy (data(:,i), k, data(:,j), k);
        h_vvc(i,j) = entropy (data(:,i), k, data(:,j), k, class, 1);
    end
end

end

end

Entropy equation implementation
function ent=entropy(vector1, k, vector2, l, vector3, m)
% (c) Karl Kuschner, College of Williamand Mary, Dept. of Physics, 2009.
%
% entropy finds all the entropies H(V), H(V,V), etc.
%
% DESCRIPTION
%     Calculates the entropy (or joint entropy if more than one argument
%     pair) of a vector (or vectors) whose values are {1,2,...k}. Must
%     send in one or more pairs of arguments ("vector", "num poss vals")
%
% USAGE
%     ent=entropy (vector, num_poss_vals [,vec,numvals [,vec,numvals] ])
%
% INPUTS One to three pairs of
%     vector: vector of integers 1,2,..k representing values of random var
%     k: number of possible values in vector
%
% OUTPUTS
%     ent: information entropy H(V1) [or H(V1,V2) or H(V1,V2,V3)]
%
% CALLED FUNCTIONS
%
%     None.

% Initialize
n=max(size(vector1)); % Number of possible cases (not error checked)

% Calculate the Entropy

% single variable entropy formula
if nargin==2
    P_k=hist(vector1,1:k)/n;
    NonZero=find(P_k~=0); % See Note 1
    ent=-sum(P_k(NonZero).*log2(P_k(NonZero)));
end

% two variable joint entropy H(V1,V2)
if nargin==4
    ent=0;
    for i=1:l
        P_lk=hist(vector1(vector2==i),1:k)/n;
        NonZero=find(P_lk~=0);
        ent=ent-sum(P_lk(NonZero).*log2(P_lk(NonZero)));
    end
end
end

```

```

% three variable joint entropy H(V1,V2,V3)
if nargin==6
    ent=0;
    for i=1:l % for all possible values in V2
        for j=1:m % for all possible values in V3
            % empirically find probability and sum entropy each
            % step
            P_lkm=hist(vector1(vector2==i & vector3==j),1:k)/n;
            NonZero=find(P_lkm~=0); % See Note 1
            ent=ent-sum(P_lkm(NonZero).*log2(P_lkm(NonZero)));
        end
    end
end

% Note 1: we can skip terms with p(a,b,c)=0 since
% p log (p) = 0 log 0 = 0
% in that case and it does not contribute to the sum.

end

Find mutual information of a vector with all columns of an array
function MIOut = MIarray(MatrixIn, class)
% by Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
%
% MIarray finds MI of each column of a data set with a separate vector
%
% DESCRIPTION
% This function finds the mutual information between a single
% discrete variable (class) and a matrix of discrete variables
% (MatrixIn) which have the same number of cases (variables in
% columns, cases in rows). A row vector containing the values
% MI(Vi,C) for each variable Vi in the matrix is returned.
%
% USAGE
% MIOut = MI(MatrixIn, class)
%
% INPUTS
% data: double array of discrete integer (1:n) values, cases in rows
% and variables in columns.
% class: double (col) vector, values 1:c representing class of each
% case. Number of values c can be different than n in the data.
%
% OUTPUTS
% MIOut: double (row) vector whose entries are the Mutual information
% between each corresponding column of MatrixIn and the class.
%
% CALLED FUNCTIONS
% None.
%
% Intialize and Data Check check arguments
if nargin~=2
    disp('wrong number of input arguments')
    disp('need (data_array,class)')
    disp(' ')
    disp('Type "doc MI" for more info')
end
%class and MatrixIn must have the same number of rows
[rows cols]=size(MatrixIn);

```

```

if size(class,2)==rows
    class=transpose(class);
elseif size(class,1)~=rows
    disp('Dimension mismatch in rows of MI arrays')
    disp('Input arrays must have the same number of rows')
    return
end %row dimension check

% States must be integer values, typically 1 to n. If so, record n.
% Similarly, find out the number of states of the class variable.
if sum(any(MatrixIn-round(MatrixIn)))
    disp('Matrix in should be integers 1 to n')
    return
else
    n=max(size(unique(MatrixIn))); % Number of data states
    c=max(size(unique(class))); % Number of class states
end % check if integer

% Variable Prep
MatrixIn=int8(MatrixIn); %optional
class=int8(class); %optional
Pcv = zeros(c,n,cols);

% Compute probability tables. P_ij is a matrix whose entries are
% Prob(Variable 1=state i and Variable 2= state j). Others are similar.

if c==1 %trap for errors in the case where allclasses are the same
    Pc = 1;
else
    % Create a 3-D array with c rows, each row filled with P(C=ci)
    Pc = repmat((hist(class,1:c)/rows)', [1,n,cols]);
end

% Create a 2-D array where (j,k) is P(Vk=vj). Replicate it to a third
% dimension to prepare for multiplication with the above.
Pv =repmat( reshape ( hist(MatrixIn,1:n)/rows , [1,n,cols] ), [c,1,1] );

% Now multiply these together, The result is a c by n by cols matrix whose
% (i,j,k) entry is P(C=ci)*P(Vk=vj) for each value of class ci and daa vj.
PcPv= Pc.*Pv;

% Now we need a similar sized array with the (i,j,k) entry equal to P(C=ci
% and Vk=vj) -- the joint probability.
for classtate=1:c
    Pcv(classtate, :, :) = hist(MatrixIn(class==classtate, :),1:n)/rows;
end

% Now we can compute the mutual info using
%
% MI(C=i,Vk=j) = sum i (sum j (Pcv(i,j,k) log [Pcv(i,j,k)/PcPv(i,j,k)] ) )
%
miterms=Pcv.*(log2(Pcv)-log2(PcPv)); % The term inside the log above...
miterms(isnan(miterms))=0; % with all the 0 log 0 entries removed

% Do the double summation and squeeze the unused dimensions
MIOut = squeeze(sum(sum(miterms,1),2))';

end

```

## Code for Creating Generated Data

The code that follows takes a real data set and creates a generated data set that models the parameters of the real data, and is suitable for input to the classifier code.

### Contents

- Find baseline values for all peaks
- Build randomized peaks from baseline values
- Designate disease class, separate subgroups
- Build diagnostic features
- Replicate and de-normalize cases

```
function [GenData, A,B] = CreateGenData (LeukData)
% (c) Karl Kuschner, College of William and Mary, Dept. of Physics, 2009.
% This function creates a generated data set with parameters modeled from
% the 2004 Leukemia data set.
%
% INPUTS
%   LeukData: Data repository structure with fields:
%       Intensities: Array of intensity values of size
%                   #cases x #variables
%       Class: Vector of length "#cases", with discrete values
%              identifying class of each case (may be integer)
%       ID: Patient ID array of length #cases, with one or more cols
%       MZ: Vector of length "#variables" holding labels for variables
%
% OUTPUTS
%   GenData : Data structure exactly like "LeukData"
%   A,B : The subgroups (class 1,2) of the Intensities matrix
```

#### Find baseline values for all peaks

```
GenData = LeukData;
int=GenData.Intensities;
intN=int(GenData.Class==1,:); clear int;
intN(intN<1)=1;
intN(:,200)=intN(:,148); %add an extra variable, copying a peak from Leuk
GenData.MZ(200)=GenData.MZ(199)+100; % Fill in an artificial m/z
mu=mean(intN);
sig=std(intN);
cv=sig./mu;
cv(cv>.2)=.2; %from data
sig=cv.*mu; clear cv;
```

#### Build randomized peaks from baseline values

A set of spectra, with the number of cases approximating the number of unique patient identification numbers in the Leukemia data, is generated via a draw from a  $N(\mu, \sigma)$  (normal) distribution for each variable individually.  $\mu$  is the value of the average spectrum at that peak position,  $\sigma$  is estimated from the Leukemia data set population. At this point there should be no real distinction between any variables.

```
for i=1:150
```

```

    GenInt(i,:) = random('normal', mu, sig);
end
clear i mu sig;

```

### Designate disease class, separate subgroups

One half of the population is designated to be in the disease class. A class vector representing this choice is created and attached to the data.

```

class(1:75)=1;
class(76:150)=2;

```

### Build diagnostic features

One feature (labeled 200) is chosen as "highly diagnostic" and the mean values of the two subpopulations (normal and disease) are separated by at two times the population's average standard deviation. Specifically, the disease cases are redrawn from  $N(\mu + 3\sigma)$ .

```

mu = mean(GenInt);
sig = .2 * mu(200);
v200(1:75) = normrnd(mu(200), sig, 75, 1);
v200(76:150) = normrnd(mu(200) + 2 * sig, sig, 75, 1); % Note mean is mu + 2sigma
v200 = v200'; figure(); bar(v200);
GenInt(:, 200) = v200;
clear mu sig

% A random fraction (about a tenth) of the total value of this
% feature is placed into each of four adjacent features (labeled 195-199).
% In this manner, five diagnostic features are created, correlated to the
% parent feature. This procedure mimics the measurement of
% adducts or modifications in the real data set, wherein slightly modified
% molecules show up as a peak separate from the original.
v196to9fact = (rand(150, 4) * .04) + .08; % 8 to 12%
for i = 1:4
    v196to9add(:, i) = v200 .* v196to9fact(:, i); %#ok<*AGROW>
end
GenInt(:, 196:199) = GenInt(:, 196:199) + v196to9add;

% Another small fraction of the value of the key peak is moved into
% a feature some distance away in the list (labeled 100), representing a
% multiply-charged ion (m/2z). This is repeated to a different feature
% (labeled 99) for one of the adducts.
v99to100fac = (rand(150, 2) * .1) + .1;
v99to100add = GenInt(:, 199:200) .* v99to100fac;
GenInt(:, 99:100) = GenInt(:, 99:100) + v99to100add;
clear v99to100add v99to100fac

% Another diagnostic feature (1.5 sigma separation) is created but
% not added to the feature list. Instead, a random amount of the total
% value of that feature (itself chosen from a normal distribution) is
% placed in two non-adjacent features (labeled 50 and 150). This
% represents the breaking apart of a biomarker protein, whose mass is too
% great to be detected, into several fragment molecules that are in the
% range of measurement.
mu = mean(GenInt(:, 50));
sig = .2 * mu;
v201(1:75) = normrnd(mu, sig, 75, 1);
v201(76:150) = normrnd(mu + 1.5 * sig, sig, 75, 1);

```

```

v201=v201';
fragfac1=(rand(150,1))*0.4;
fragfac2=1-fragfac1;
v50add=v201.*fragfac1;
v150add=v201.*fragfac2;
GenInt(:,50)=GenInt(:,50)+v50add;
GenInt(:,150)=GenInt(:,50)+v150add;
clear fragfac1 fragfac2 mu sig v150add v50add v201

```

Two more features (labeled 1 and 2) are selected as "moderately diagnostic" and the values chosen from two normal distributions whose means are separated by about two standard deviations of either group. Specifically, the disease cases are redrawn from  $N(\mu+1.5\text{sig}, \text{sig})$ . One of these two features has a portion of the other feature's value added to it to represent an unsuccessful deconvolution of two peaks that are so close together the peak value of one is "riding up" on the tail of another. Two mildly diagnostic features (3 and 4) are created without this effect.

```

for i=1:2
    mu=mean(GenInt(:,i));
    sig=.2*mu;
    GenInt(1:75,i)=normrnd(mu,sig,75,1);
    GenInt(76:150,i)=normrnd(mu+1.5*sig,sig,75,1);
end
shoulder=rand(150,1)*0.1+0.1; % 10-20%
GenInt(:,1)=GenInt(:,1)+shoulder.*GenInt(:,2);
for i=3:4
    mu=mean(GenInt(:,i));
    sig=.2*mu;
    GenInt(1:75,i)=normrnd(mu,sig,75,1);
    GenInt(76:150,i)=normrnd(mu+1*sig,sig,75,1);
end
clear i mu sig shoulder

```

### Replicate and de-normalize cases

The cases are replicated three times (the original of each case is discarded) by multiplying each value by normalization factor. For a single data vector  $X$  a factor  $f$  is first selected from  $\sim U(0.5, 2.0)$  to replicate the range of total ion current normalization factors found in the Leukemia data.

```

for i=1:150;
    for j=1:3
        casenum=(i-1)*3+j;
        % thiscase=GenInt(i,:);
        basefactor = rand*1.5+.5;
        % factorvec=rand(1,200)*0.1+basefactor;
        FinInt(casenum,:) = GenInt(i,:)*basefactor;
        FinCl(casenum)=class(i);
        FinID(casenum,1)=i; %patient ID
        FinID(casenum,2)=j; %replicate number
    end
end % de-normalization
figure();imagesc(log(GenInt));
GenData.Intensities=FinInt;
GenData.ID=FinID;
GenData.Class=FinCl';
Int=GenData.Intensities;
A=Int(1:225,:);

```

```
B=Int(226:450,:);clear Int;  
end %of CreateGenData function
```



## APPENDIX C: RESULTS

This section lists the detailed results for the three data sets under repeated application of the naïve Bayesian classifier.

### Generated Data

The error rates for the forward selection of the generated data features are listed below. Each trial is a column listing the error rate resulting from a feature set size as listed in the leftmost column.

Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10
1	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%	16.6%
2	12.5%	12.5%	12.4%	12.5%	12.5%	12.5%	12.5%	12.5%	12.4%	12.3%
3	9.9%	9.8%	9.9%	9.9%	9.7%	9.9%	9.9%	9.6%	10.0%	9.8%
4	8.1%	8.2%	8.1%	8.1%	8.3%	8.0%	8.1%	8.2%	8.1%	8.1%
5	7.6%	6.9%	7.0%	7.5%	7.1%	7.5%	7.1%	7.0%	7.0%	6.9%
6	6.6%	5.0%	6.1%	6.4%	5.2%	6.8%	5.1%	5.4%	5.3%	5.2%
7	5.7%	4.1%	5.4%	5.7%	4.2%	5.9%	4.0%	4.2%	4.2%	4.1%
8	5.0%	3.3%	4.9%	5.4%	3.5%	5.0%	3.5%	3.4%	3.5%	3.5%
9	4.1%	2.8%	4.6%	4.9%	2.9%	4.7%	2.8%	3.0%	3.1%	3.0%
10	3.3%	2.8%	4.3%	4.5%	2.6%	4.6%	2.9%	2.8%	2.8%	2.7%
11	2.7%	2.6%	3.8%	4.3%	2.5%	4.7%	2.6%	2.7%	2.7%	2.5%
12	2.3%	2.0%	3.8%	4.0%	2.4%	4.4%	2.3%	2.4%	2.6%	2.6%
13	2.1%	1.9%	3.6%	3.8%	2.0%	3.8%	2.2%	2.0%	2.5%	2.0%
14	1.7%	1.7%	3.5%	3.8%	1.9%	3.6%	2.1%	1.9%	2.5%	1.9%
15	1.6%	1.7%	3.6%	3.0%	1.9%	3.4%	2.2%	1.9%	2.3%	1.9%
16	1.5%	1.5%	3.5%	2.5%	2.0%	3.5%	2.1%	1.8%	2.1%	1.8%
17	1.2%	1.5%	3.4%	2.3%	1.7%	3.5%	2.0%	1.8%	2.0%	1.7%
18	1.1%	1.4%	3.3%	2.2%	1.7%	3.5%	1.9%	1.5%	1.9%	1.7%
19	1.0%	1.4%	3.0%	2.1%	1.7%	3.6%	1.7%	1.5%	1.5%	1.6%
20	1.0%	1.2%	3.0%	2.0%	1.7%	3.6%	1.6%	1.2%	1.3%	1.8%

## Leukemia Data

The following table lists the first twenty features from the Leukemia data set selected via forward selection in each of ten trials.

Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10
1	199	199	199	199	199	199	199	199	199	199
2	198	198	198	198	198	198	198	198	198	198
3	141	141	141	141	141	193	141	141	141	141
4	43	43	43	43	122	18	80	122	43	122
5	31	122	4	4	9	93	122	9	4	9
6	90	151	18	27	102	122	193	102	122	193
7	102	76	122	44	10	151	151	10	151	151
8	122	118	151	18	5	118	116	5	76	120
9	151	27	10	142	191	71	173	105	118	41
10	68	68	74	42	109	76	119	151	55	173
11	93	109	11	3	193	11	30	193	38	27
12	104	70	158	151	151	54	5	182	179	133
13	39	62	68	193	110	48	10	136	3	99
14	169	33	171	1	68	86	68	71	144	182
15	120	193	39	10	71	128	25	48	74	186
16	3	172	193	68	4	33	104	24	68	38
17	110	158	5	171	86	44	38	70	135	25
18	167	3	53	186	118	56	3	192	193	3
19	191	115	188	30	39	158	56	39	30	188
20	38	38	179	38	101	148	27	68	168	26

The error rates associated with the selections listed above are shown in the table below.

Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10
1	17.2%	17.2%	17.1%	17.2%	17.1%	17.1%	17.2%	17.2%	17.2%	17.1%
2	12.6%	12.4%	12.6%	12.6%	12.4%	12.5%	12.5%	12.5%	12.5%	12.4%
3	9.7%	10.0%	9.9%	9.9%	9.8%	10.0%	9.9%	9.8%	9.7%	10.0%
4	8.6%	8.7%	8.6%	8.6%	8.7%	8.7%	8.8%	8.7%	8.8%	8.7%
5	8.2%	8.2%	8.2%	8.1%	7.5%	7.9%	8.0%	7.6%	8.3%	7.3%
6	7.9%	7.1%	7.8%	7.9%	6.6%	7.2%	7.1%	6.7%	7.8%	6.5%
7	7.6%	6.3%	7.5%	7.7%	6.0%	6.1%	6.3%	6.2%	6.7%	6.0%
8	7.3%	5.8%	6.3%	7.5%	6.1%	5.2%	5.6%	6.1%	5.9%	5.6%
9	6.2%	5.4%	5.7%	7.2%	5.9%	5.0%	5.2%	6.0%	5.6%	5.1%
10	5.8%	5.2%	5.5%	6.8%	5.9%	4.5%	4.9%	5.9%	5.4%	4.5%
11	5.4%	5.1%	5.3%	6.3%	5.8%	4.1%	4.7%	5.2%	5.4%	4.4%
12	5.3%	4.9%	5.3%	6.0%	5.4%	4.1%	4.5%	4.6%	5.0%	4.2%
13	5.1%	5.0%	5.2%	5.3%	5.0%	3.9%	4.3%	4.3%	4.9%	4.5%
14	5.1%	5.0%	5.0%	5.0%	4.7%	4.0%	4.2%	4.3%	4.2%	4.4%
15	4.8%	4.9%	4.8%	4.7%	4.5%	3.9%	4.1%	4.2%	4.1%	4.4%
16	4.7%	4.7%	4.7%	4.6%	4.3%	3.8%	3.9%	4.2%	3.9%	4.3%
17	4.5%	4.5%	4.5%	4.6%	4.3%	3.9%	4.1%	4.2%	3.9%	4.0%
18	4.3%	4.5%	4.4%	4.3%	4.3%	3.7%	4.0%	4.3%	3.7%	3.8%
19	4.2%	4.4%	4.3%	4.4%	4.1%	3.8%	3.6%	4.3%	3.6%	3.4%
20	4.0%	4.1%	4.1%	4.3%	4.1%	3.7%	3.4%	4.3%	3.5%	3.4%

## PCA Data

The next table gives the first thirty features selected from the PCA data set in each of ten trials.

Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10
1	71	71	71	71	71	71	71	71	71	71
2	1	1	1	1	1	1	1	1	1	1
3	9	72	77	9	77	77	77	9	77	77
4	23	77	37	23	37	37	37	23	37	37
5	99	37	75	99	75	75	75	73	75	75
6	38	25	65	73	83	65	84	77	84	25
7	73	75	19	38	25	83	65	91	25	83
8	77	84	84	80	6	81	81	66	66	66
9	4	38	38	17	66	19	68	38	19	91
10	19	81	66	58	38	66	19	27	85	81
11	57	64	25	25	9	25	38	48	95	19
12	49	6	2	77	57	6	58	6	11	9
13	84	9	11	91	91	9	6	62	9	58
14	78	57	52	84	84	38	46	75	38	38
15	54	3	72	78	7	11	9	20	58	20
16	80	52	6	6	58	57	57	41	6	64
17	6	86	78	3	81	3	17	7	46	6
18	66	19	3	20	78	2	3	25	73	52
19	58	11	5	33	20	72	78	83	80	86
20	91	78	99	57	46	91	90	47	83	90
21	25	74	81	56	97	14	69	81	23	74
22	74	21	20	8	19	80	25	11	31	62
23	83	20	57	31	23	64	83	5	8	47
24	75	98	12	75	73	52	47	37	7	89
25	20	79	83	10	5	82	63	58	90	78
26	10	17	21	64	17	29	5	31	29	21
27	60	43	17	81	45	16	74	65	99	60
28	17	58	80	54	68	7	54	84	43	11
29	90	48	93	4	13	31	12	33	17	17
30	81	29	31	37	3	73	20	57	91	41

The final table lists the error rate associated with the selections listed above.

Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10
1	38%	38%	38%	38%	38%	38%	38%	38%	38%	38%
2	37%	37%	37%	37%	37%	37%	38%	37%	37%	37%
3	37%	37%	37%	37%	37%	37%	37%	37%	37%	37%
4	36%	38%	35%	36%	35%	34%	35%	36%	35%	34%
5	35%	35%	34%	35%	34%	34%	34%	35%	34%	34%
6	34%	34%	33%	34%	33%	33%	34%	33%	33%	34%
7	32%	33%	33%	33%	32%	33%	32%	32%	32%	32%
8	32%	32%	32%	32%	32%	31%	32%	31%	31%	31%
9	31%	32%	31%	32%	31%	31%	31%	30%	31%	31%
10	30%	31%	31%	32%	30%	31%	30%	30%	30%	30%
11	30%	31%	30%	31%	30%	30%	31%	30%	30%	30%
12	30%	30%	30%	31%	29%	30%	30%	30%	30%	30%
13	30%	29%	30%	30%	28%	30%	30%	29%	31%	29%
14	30%	28%	30%	30%	29%	29%	30%	29%	31%	29%
15	30%	28%	31%	29%	29%	29%	29%	29%	31%	29%
16	30%	28%	31%	29%	29%	29%	29%	29%	31%	28%
17	30%	28%	30%	28%	28%	29%	29%	29%	30%	28%
18	30%	29%	30%	28%	28%	30%	29%	28%	30%	28%
19	29%	28%	30%	29%	28%	30%	29%	28%	30%	28%
20	29%	29%	30%	29%	28%	30%	29%	28%	30%	29%
21	29%	29%	30%	29%	28%	30%	30%	29%	29%	29%
22	29%	29%	30%	29%	28%	30%	30%	29%	28%	29%
23	29%	29%	30%	30%	29%	30%	30%	30%	28%	30%
24	29%	29%	30%	30%	28%	30%	30%	30%	28%	30%
25	28%	30%	30%	30%	28%	30%	30%	30%	28%	30%
26	28%	30%	31%	30%	28%	30%	30%	30%	28%	30%
27	28%	30%	30%	29%	28%	31%	30%	30%	28%	30%
28	29%	30%	31%	30%	28%	31%	31%	30%	29%	31%
29	29%	30%	31%	30%	29%	31%	31%	30%	29%	31%
30	29%	30%	31%	30%	29%	31%	31%	30%	29%	31%

## INDEX

- Adducts and modifications
  - combination into metavariables, 92
- Adjacency matrix, 82
- Approximate searches, 50
- Background Subtraction, 29
- Bayes' Theorem
  - description, equation, 13
  - example, 15
  - prior, 15, 40
- Bayesian network
  - algorithm, 76
  - and causality, 47
  - definition, 42
- Bayesian network structures
  - diverging (inverted V) connection, 45
  - isolated node, 46
  - number of possible DAGs, 49
  - serial connection, 44
  - triply connected structure, 76
  - V connection, 45
- Bias avoidance, 17
- Binning, bin. *See* Discretization
- Biomarkers, 4
- Classifier
  - construction, 34
  - deterministic vs. probabilistic, 35
- Conditional probabilities, 8
- Cross-validation
  - general, 56
  - leave one out method, 57
  - methods, 57
  - n-fold, 58, 62
  - stratification, 59
- Deconvolution, 29
- Directed acyclic graph
  - ancestors and descendants, 43
  - DAG, 42
  - encoding joint probability distributions, 44
  - example, 42
  - parent and child, 43
  - representing a probability distribution, 43
- Discretization, 55, 83
  - boundary optimization, 86
  - naïve, 83
  - optimized, 83
- Entropy
  - conditional, 10
  - definition, 9
  - equation, 10
  - information vs. thermodynamic, 9
  - joint entropy, 10
  - maximum and minimum, 10
  - Shannon, 9
- Error rate
  - cross-validated, 58
- Feature selection
  - backward elimination, 37, 64
  - definition, 34
  - filter and wrapper techniques, 36
  - forward selection, 37, 64, 97
  - pseudo-code, 37
- Independence
  - condition, 40
  - problem caused by, 41
- Instantiation, 45
- Instrument function, 53
- Marginalization, 14
- Markov blanket, 51, 75
- Mass spectrometry, 1
- Metavariables, 91
- Multiply charged ion, 48
- Mutual information, 11
  - as a function of entropy, 12
  - between variables and the class, 54
  - conditional, 13
  - equation, 11
  - maximum and minimum, 12
  - threshold, 55, 82
  - threshold factor, 82
  - used for structure learning, 79
- Naïve Bayesian classifier, 38, 60
- Normalization
  - by total ion current, 31
- Parameter learning, 51
  - use of maximum likelihood, 52
- Peaks, 2
- Probability
  - conditional, 7
- Probability distribution function (PDF), 8
- Probability tables, 51, 77, 93
- Product Rule, 8
- Quality control, 18
- Replicates, 19, 20, 33
  - replicate averaging, 29
- Satellite. *See* Multiply charged ion
- Structure learning, 49
  - approaches, 77
  - classification tree, 78
  - first level, 83
  - maximum weight spanning tree, 78
  - parent-child identification, 88
  - triply-connected structures, 87
- Sum Rule, 8
- Training set, 38

## GLOSSARY

**Adjacency matrix** – A square matrix of logical (true or false) values. The entry at the row  $i$  and column  $j$  position represents the truth value of the statement “in the Bayesian network represented by this adjacency matrix, there is a directed arc between node  $i$  and node  $j$ .”

**Cross-validated error rate** – The error rate that is found by first classifying a sub-population based on parameters derived from the remainder of the population, repeating this process until all samples in the total population have been classified, then comparing each sample’s predicted class to its known class.

**Deterministic Classifier** – A classifier that assigns a case to a specific class, rather than assigning the probability of being in a class.

**False Positive/False Negative** – A classifier error where a case is deterministically labeled with the wrong class. In this document, a false positive refers to a classification that a disease is present, when the clinical diagnosis was that a disease was not present. A false negative refers to a classification that the case was normal, when the clinical diagnosis was that a disease was present.

**Instantiation** – A node in a Bayesian network is instantiated when its value becomes known. This knowledge is propagated throughout the network, and the probability tables of nodes whose values are unknown are adjusted.

**Naïve binning** – Discretizing data based on parameters derived from the entire population. For example, a continuous variable can be discretized by the test  $b=0$  if  $x > \mu$ ,  $b=1$  otherwise, where  $x$  is the continuous value,  $b$  is the discrete value, and  $\mu$  is the population mean.

**Probabilistic Classifier** – a classifier whose output is the probability (from 0 to 1) that the case being classified is one of a set of possible class outcomes.

**QC Spectra** – Set of spectra created from a mixed pool of blood sera from many people, then sampled many times. Variations in the results are assumed to represent the variability of the preparation and measurement processes, rather than the population.

**Replicates** – A blood serum sample from a single patient is divided into several samples and the entire process of chemical preparation, mass spectrometry, and signal processing is completed independently on each. The resulting data is called a replicate and may be treated independently, or averaged with other replicates to reduce experimental variation.

**Markov blanket** – In a Bayesian network, the Markov blanket of a node  $N$  is the minimum set of other nodes  $S$  such that if all the nodes in  $S$  are instantiated, the probability of  $N$  is fixed, regardless of the values of nodes not in  $S$ . Once the Markov blanket is determined, additional structures are irrelevant to  $N$ , and can be ignored if  $N$  is the only node of interest.

## WORKS CITED

- [1] **National Cancer Institute.** *The Early Detection Research Network Fourth Report.* Bethesda, MD : National Institute of Health, 2008.
- [2] **Malyarenko, Dariya.** *Personal Interview.* June 2008.
- [3] **Rifai, Nader and Gillette, M. A. Carr, Steven.** *Protein biomarker discovery and validation: the long and uncertain path to clinical utility.* 24, 2006, *Nature Biotechnology*, pp. 971-983.
- [4] **Jaynes, E. T.** *Probability Theory: the logic of science.* Cambridge : University Press, 2003. pp. 24-34. 0-521-59271-2.
- [5] **Shannon, Claude E.** *A Mathematical Theory of Communication.* October, s.l. : Bell Systems Technical Journal, 1948, Vol. 27.
- [6] **Cover, Thomas M. and Thomas, Joy A.** *Elements of Information Theory.* s.l. : John Wiley & Sons, Inc, 1991. 0-471-06259-6.
- [7] **Baggerly, Keith A., Coombes, Kevin R. and Morris, Jeffrey S.** *Bias, Randomization, and Ovarian Proteomic Data: A Reply to "Producers and Consumers".* 1, 2002, *Cancer Informatics*, Vol. 1.
- [8] **Semmes, O. J., et al.** *Discrete protein signatures discriminate between adult T-cell leukemia and HTLV-1-associated myelopathy.* 19, 2005, *Leukemia*.
- [9] Definition of Smoldering Leukemia. *Dictionary of Cancer Terms.* [Online] National Cancer Institute. [Cited: September 3, 2008.] [http://www.cancer.gov/Templates/db\\_alpha.aspx?CdrID=46583](http://www.cancer.gov/Templates/db_alpha.aspx?CdrID=46583).
- [10] Myelodysplastic syndromes. *Disease Information.* [Online] Leukaemia Research Foundation. [Cited: September 3, 2008.] <http://www.lrf.org.uk/en/1/infdispatmye.html>.
- [11] **Gatlin-Bunai, Christine L., et al.** *Optimization of MALDI-TOF MS Detection for Enhanced Sensitivity of Affinity-Captured Proteins Spanning a 100 kDa Mass Range.* 6, 2007, *Journal of Proteome Research*, pp. 4517-4524.
- [12] **Rasmussen, G. T. and Isenhour, T. L.** *The Evaluation of Mass Spectral Search Algorithms.* 3, 1979, *J. Chem. Inf. Comput. Sci.*, Vol. 19, pp. 179-186.
- [13] **Weaver, Dennis.** *Analysis of QC spectra correlations.* 2008. Internal Report.



- [14] **Oh, Jung Hun, et al.** *Diagnosis of Early Relapse in Ovarian Cancer Using Serum Proteomic Profiling*. 2, 2005, Genome Informatics, Vol. 16.
- [15] **Miller, Alan J.** *Subset Selection in Regression*. 2d ed. Boca Raton, FL : CRC Press, 2002. pp. 39-45. ISBN 1-58488-171-2.
- [16] **Langley, Pat and Sage, Stephanie.** *Induction of Selective Bayesian Classifiers*. Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence. Seattle, WA, 1994.
- [17] **Jensen, Finn V. and Nielson, Thomas D.** *Bayesian Networks and Decision Graphs*. New York, NY : Springer, 2007. ISBN 0-387-68281-3.
- [18] **Pearl, Judea.** *Causality*. Cambridge, UK : Cambridge University Press, 2000. ISBN 0-521-77362-8.
- [19] **Robinson, R. W.** Counting unlabeled acyclic digraphs. [ed.] C. H. C. Little. *Combinatorial Mathematics V*. Berlin : Springer, 1977, Vol. 622, pp. 28-43.
- [20] **Heckerman, David.** *A Tutorial on Learning With Bayesian Networks*. Redmond Washington : Microsoft Research, 1995. MSR-TR-95-06.
- [21] **Marchetelli, Robert.** *Analysis of Quality Control Data*. Department of Physics, College of William and Mary. Williamsburg, VA : s.n., 2005. Internal Report.
- [22] **Elisseeff, M. and Ponti, A.** *Leave-one-out error and stability of learning algorithms with applications*. [ed.] J. Suykens. s.l. : IOS Press, 2002, NATO-ASI Series on Learning Theory and Practice.
- [23] **Kohavi, Ron.** *A Study of Cross Validation and Bootstrap for Accuracy Estimation and Model Selection*. Stanford, 1995.
- [24] **Chow, C. and Liu, C.** *Approximating discrete probability distributions with dependence trees*. 1968, IEEE Transactions on Information Theory, Vol. 14, pp. 462-467.
- [25] **LeRay, Philippe and Francois, Olivier.** *BNT Structure Learning Package: Documentation and Experiments*. Rouen, France : Laboratoire PSI - Universite et INSA de Rouen, 2004.
- [26] **Cheng, Jie, et al.** *Learning Bayesian networks from data: An information-theory based approach*. 2002, Artificial Intelligence, Vol. 137.
- [27] **Chickering, David Maxwell and Meek, Christopher.** *Monotone DAG Faithfulness: A Bad Assumption*. Microsoft Research. Redmond, WA : Microsoft Corporation, 2003.

- [28] **Mann, Mathias, Hendrickson, Ronald C. and Pandey, Akhilesh.** *Analysis of Proteins and Proteomes by Mass Spectrometry.* 70, 2001, *Annu. Rev. Biochem.*, pp. 437-73.
- [29] **Trosset, Michael W.** *An Introduction to Statistical Inference and its Applications.* Williamsburg, VA : Unpublished, 2006.
- [30] **Hortin, Glen L.** *The MALDI-TOF Mass Spectrometric View of the Plasma Proteome and Peptidome.* 52, Jul 2006, *Clin. Chem.*, pp. 1223-1237.
- [31] **Tolson, Jonathon and al., et.** *Serum protein profiling by SELDI mass spectrometry: detection of multiple variants of serum amyloid alpha in renal cancer patients.* 84, 2004, *Laboratory Investigation*, pp. 854-856.

## VITA

Karl Kuschner was born in northern Virginia and grew up in Melbourne, Florida. After graduation from Eau Gallie High School in 1978, he entered the United States Air Force Academy in Colorado Springs, Colorado. He graduated with a Bachelor of Science in Physics and Mathematics, and was commissioned as an officer in the United States Air Force on June 2, 1983.

He attended Undergraduate Pilot Training and eventually was stationed in Germany as a pilot of the F4-E. Over the next eight years, he was stationed around the world and won several decorations for valor in combat. He entered the Graduate School of the University of Texas at Austin, Texas in August 1992. After receiving a Master of Arts degree in Physics, he returned to the cockpit and spent the remainder of his Air Force career as an F-4G Wild Weasel and F-15C Eagle pilot, serving in multiple combat zones and other worldwide deployments. He also received a Master of Science degree in National Security Studies from the U.S. Naval War College in Newport, Rhode Island.

He retired from the Air Force in 2004 and began his graduate program in the Department of Physics at the College of William and Mary in Williamsburg, Virginia.

He is married to Bjork Jonasdottir and has two children.