OPTIMIZATION OF PRE-PROCESSING VARIABLES FOR HYPERSPECTRAL ANALYSIS OF FOCAL PLANE ARRAY FOURIER TRANSFORM INFRARED IMAGES

By

Tommy Pinchuk

Department of Food Science and Agricultural Chemistry
McGill University, Montreal, Canada

April, 2006

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Masters of Science.

Your file Votre référence
ISBN: 978-0-494-24770-9
Our file Notre référence
ISBN: 978-0-494-24770-9

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to start off thanking my parents – for without them I would not be able to participate in the quality education I am privileged to have. My parents along with my sister have always been there for support, encouragement and understanding with this endeavor.

I like to thank two of my peers - Mr. Jonah Kirkwood for graciously providing me with excellent image data, and Mr. Andrew Ghetler for his relentless brainstorming, continuous input, feedback and revisions.

I appreciate the love and support of my wife Tanya as well as her understanding in delaying our honeymoon so that I may finish this project. I would also like to thank her for her input, interest and editing revisions.

Working with me to the final hour, I appreciate the time and effort of Jacqueline Sedman for her profession revisions and feedback.

Most of all, I would like to express my gratitude to Dr. Ashraf Ismail for his leadership, guidance and supervision in my research pursuit. His passion and persistence have been a driving factor in my accomplishment. I would also like to express my thanks to the rest of the food science faculty and McGill University for their resources and attention.

Abstract

A genetic algorithm was employed to select the optimal combination of pre-processing variables, including data pretreatment, data manipulation and feature extraction procedures, for eventual clustering of a data set consisting of hyperspectral images acquired by a focal plane array Fourier transform infrared (FPA-FTIR) spectrometer. The data set consisted of infrared images of bacterial films, and the classification task investigated was the discrimination between Gram-positive and Gram-negative bacteria. The genetic algorithm evaluated combinations of variables pertaining to bacterial film thickness tolerances, baseline correction, pixel co-addition, outlier removal, smoothing, mean centering, normalization, derivatization, integration and principal component selection. Following numerous iterations of unsupervised processing, the genetic algorithm arrived at a sub-optimal solution yielding a clustering accuracy of 97.8% and a data utilization of 28.6%. The results provided insight into the co-dependencies of the pre-processing variables and their consequential effect on the selected data. The robustness of the classification model was evaluated and reinforced by the successful classification of two distinct validation sets. The overall success of the genetic algorithm suggests that it is an effective time saving resource for the optimization of pre-processing variables that does not require operator intervention.

Résumé

Un algorithme génétique a été employé pour choisir la combinaison optimale des variables de prétraitement comprenant la préparation des données, le traitement des données et les procédures d'extraction de traits significatifs afin de grouper des images hyperspectrales acquises par un spectromètre infrarouge à transformée de Fourier utilisant un détecteur matriciel au plan focal (FPA-FTIR). L'ensemble de données était composé des images infrarouges de films bactériens, and la tâche de classification était la discrimination entre des bactéries gram positif et des bactéries gram négatif. L'algorithme génétique évalue les combinaisons de variables concernant des tolérances d'épaisseurs de films bactériens, la correction des lignes de base, l'ajoutement des points, l'enlèvement des points discordants, le lissage, la normalisation, la dérivation, l'intégration et l'analyse en composantes principales. Après de nombreux traitements non supervisés, l'algorithme génétique est arrivé à une solution optimale comportant une exactitude de 97,8% avec une utilisation des données de 28,6%. Les résultats fournis précisent les interdépendances des variables de prétraitement et de leurs effets sur les données. La fiabilité du modèle de classification a été évaluée et renforcée par la classification réussie de deux différents ensembles. Le succès de l'algorithme génétique démontre le gain de temps pour l'optimisation des variables de prétraitement sans la nécessité d'un opérateur.

1 Introduction

Modern chemical and spectral analysis techniques are fundamentally based on the mathematical manipulation of experimental data (Adams, 1995). The term *chemometrics* was proposed more that 20 years ago to describe the mathematical manipulation techniques and operations associated with the interpretation of chemical data. In general, chemometric analysis is applied to determine either the quantitative composition of a sample or the qualitative classification of a species (Adams, 1995). It is essential that analysts comprehend how their data is obtained, modified and transformed to produce the information that they require.

Chemometrics has played a major role in the development of new analytical applications of spectroscopic techniques, particularly near- and mid-infrared spectroscopy, over the past few decades. The rapid growth in popularity of these techniques was triggered by the availability of laboratory computers, which allowed the large amount of data that these techniques provide to be accessed directly, but this in turn required the development of means of manipulating these data to extract relevant and reliable information. The resulting advances in chemometrics as applied to infrared spectroscopic data have extended the scope of infrared spectroscopic analysis beyond the traditional realm of chemistry. An example of particular relevance to the research reported in this thesis is the utilization of infrared spectroscopy in microbiological analysis (Naumann, 2000).

With continuing advances in instrumentation, such as the development of infrared imaging technologies, further recourse to chemometrics is needed to handle the increasingly complex tasks of data manipulation. For instance, a Fourier transform

infrared (FTIR) spectrometer equipped with an $n \times n$ focal-plane-array (FPA) detector collects an image consisting of n^2 individual spectra, where each spectrum provides information specific to a particular location in the sample. Analysis of the resulting data, termed hyperspectral data analysis, is inherently complex, given the impossibility of visualizing the full spatial and spectral information content of the acquired image on a three-dimensional plot.

The overall objective of the research presented in this thesis was to determine the optimum methods of pretreatment, preprocessing and eventual clustering of a data set consisting of infrared images acquired by a FPA-FTIR spectrometer. The literature review section of this thesis will present a brief overview of elementary statistics and infrared spectroscopic acquisition. The discussion will explore several common methods used to process infrared spectral data, as well as methods of selecting and extracting spectral features relative to clustering and classification. The literature review will also address the fundamental theories behind genetic algorithms as a method of evaluating different combinations of pre-processing variables in order to determine the optimal combination for data classification.

Chapter 3 will explore and evaluate the use of genetic algorithms to optimize the combination of data pretreatment, data manipulation and feature extraction procedures to effectively cluster Gram-positive and Gram-negative bacterial species based on their spectral profiles. The choice of this data set is related to ongoing research on the potential utility of FPA-FTIR spectroscopy in the identification of bacteria (Kirkwood et al., 2004) but may be regarded as arbitrary in the context of the present study. Data pretreatment procedures such as co-addition, outlier removal, spectral quality assessments, and

baseline correction will be evaluated for their impact on the data clustering. Data manipulation techniques such as spectral smoothing, mean centering, normalization, derivatization and integration will also be evaluated. Feature extraction techniques such as principal component analysis will be tested as well. The effectiveness of the genetic algorithm, in combination with a good fitness function, in selecting the optimal combination of pre-processing variables will then be evaluated using two independent validation tests. Overall conclusions of this study will be presented in Chapter 4.

2 Literature Review

The popularity of interdisciplinary studies has paved the way to the discovery of new methods and procedures for chemical and biological studies. The fusion of mathematics, computer science, and chemistry has given birth to the chemometrics discipline. In turn, the application of chemometrics in conjunction with infrared spectroscopic methods of analysis, traditionally associated with the field of chemistry, has provided new methods for the analysis of biological samples. For example, the research reported in this thesis was undertaken in relation to the application of infrared spectroscopy as a tool for the classification of bacteria, and thus the first section of this chapter documents the microbiological task of Gram-classification. Subsequent sections reference some elementary statistical concepts essential to the comprehension and implementation of mathematical modeling. The process of spectral acquisition is then explored as well as the techniques used to manipulate the data, select features and extract them for purposes of sample classification. The classification technique used in this work is documented as well. Finally, the principles of the functionality of a genetic algorithm are explained including the reproduction and fitness functions.

2.1 Gram-Taxonomy

The cell wall of a bacterium is perhaps one of its most distinguishing features. Not only does the cell wall structurally maintain the bacteria but it also helps to maintain the cell's characteristic shape, counter the effects of osmotic pressure and provide characteristics for viral susceptibility (UOT, 1995). The composition of the cell-wall is one of the primary characteristics analyzed in bacterial species differentiation.

In 1884 the Danish Physician Hans Christian Gram, developed a procedure for staining bacteria (Firkin and Whitworth, 1987). This procedure is known today as the Gram staining procedure. Later, this procedure would become the benchmark for bacterial classification on the basis of the cell wall permeability.

Gram staining is a relatively simple laboratory procedure as illustrated in Figure 2-1. The procedure consists of obtaining a cultured bacterial specimen and smearing it on a slide. The slide is then subjected to Crystal violet stain for 10 seconds and rinsed with water. It is then flooded with iodine for 10 seconds and rinsed again. Immediately the slide is rinsed with a decolorization solution of 95% ethanol until the thinnest part of the smear become colorless (Hashimoto and Birch, 1996).

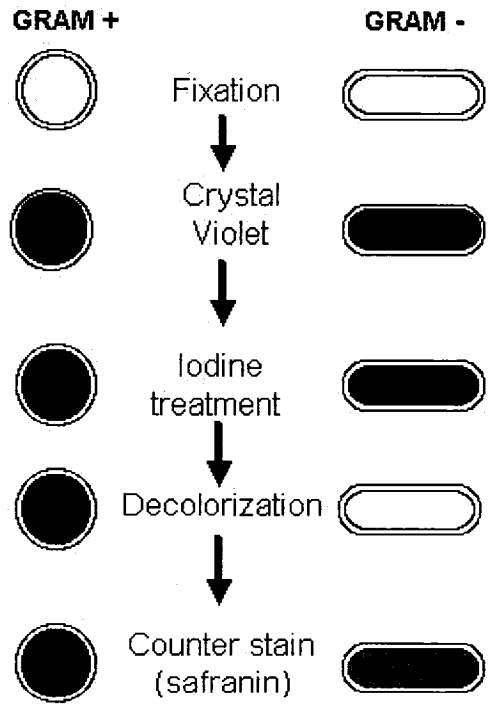

Figure 2-1: The Gram-staining procedure

The cell wall of Gram-positive bacteria is relatively thick because it consists of many layers of the polymer peptidoglycan (UOT, 1995). Therefore, if the bacterial smear

is Gram-positive it will retain purple iodine dye complexes after the de-colorization procedure is complete (Hashimoto et al., 1996).

Conversely, the cell wall of Gram-negative bacteria is relatively thin. The purple iodine dye vanishes during the de-colorization procedure (UOT, 1995). To identify if Gram-negative bacteria exist on the sample slide, the smear is flooded with safranin for 10 seconds and allowed to air dry, leaving the Gram-negative bacteria with a pink colorization as illustrated in Figure 2-1 (Hashimoto et al., 1996).

Today, the classification of unidentified bacteria into Gram-positive or Gram-negative categories is an essential analytical technique spanning many disciplines. Physicians can make a presumptive etiologic diagnosis of bacterial meningitis, bacterial pneumonia, bacteriuria, gonorrhea, and pyogenic infections of the brain, lung, abdomen, pelvis and wounds or early clinical decisions based on the examination of a Gram stained smear of infectious material (Hashimoto et al., 1996). Food scientists working with meat production can develop adequate measures for preventing spoilage and identifying potentially harmful toxins (Davies and Board, 1998). Microbiologists and immunologists can determine the adequacy of a specimen for culture and further examination using a Gram test as opposed to wasting their time and financial resources. (Hashimoto et al., 1996).

2.2 Statistical Concepts

Any method of chemometric evaluation of spectral data is derived from mathematical statistical analysis. For instance, the classification of a sample by comparison of its infrared spectrum with a standard set of spectra in a pre-recorded database involves some degree of quantitative measure of similarity in order to determine

the best match (Adams, 1995). In order to comprehend these methods, it is important to understand some key statistical concepts with pertinence to chemometric analysis.

2.2.1 Gaussian Distribution

The Gaussian distribution is the most important distribution for continuous data because of its range of practical applications in spectral analysis (Adams, 1995). The Gaussian function represents the distribution of truly random phenomena. The most common illustration of a Gaussian curve is rolling several dice many times and recording the output. The more dice that are rolled, and the more times the dice are rolled, the more the function becomes continuous and resembles the Gaussian curve as illustrated in Figure 2-2 (Kowaski et al., 1986).

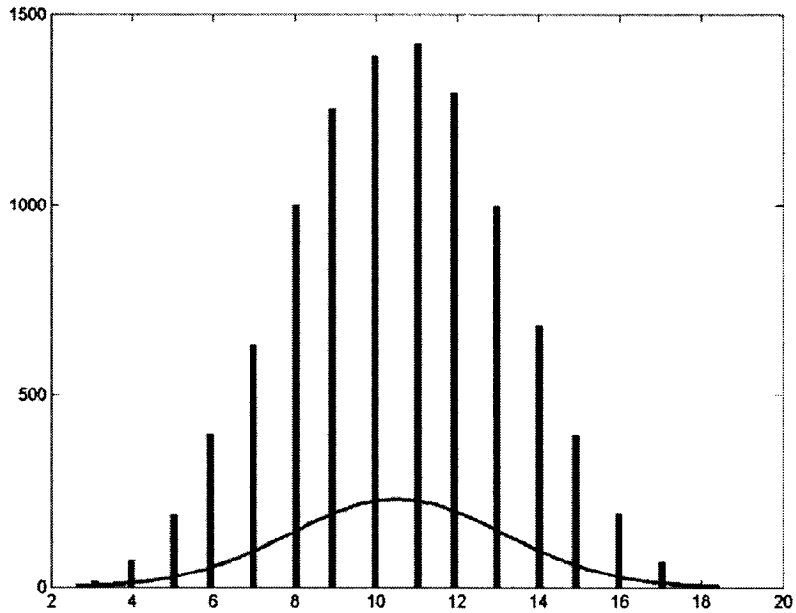

Figure 2-2: The sum of three dice being rolled 10,000 times

The continuous normal distribution (the red line in Figure 2-2) is normally adjusted so that the area under the curve is equivalent to unity or 1. The equation for the

continuous normal distribution is given by the Gaussian function in Equation 2-1 (Adams, 1995). In practice, however, only a finite number of samples exist, and therefore the Gaussian function is also represented as in Equation 2-2 (Adams, 1995).

Equation 2-1

$$f(x) = \frac{1}{\sigma\sqrt{(2\pi)}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]$$

Equation 2-2

$$f(x) = \frac{1}{s\sqrt{2\pi}} \exp\left[\frac{-(x-\bar{x})^2}{2s^2}\right]$$

$$\bar{x} = \sum_{i=1}^{n} x_i / n$$

$$s^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{(n-1)}$$

where
$f(x)$ is the height of the curve at some value x.
μ, $\bar{x}$ is the mean or average value of the function.
σ^2, s^2 is the variance.
σ, s is the standard deviation.
n is the total number of samples.
i denotes the individual elements of the set of data.

The standard deviation is a measure from the center or mean of the Gaussian curve. In practice, less than 1 in 3 of the samples will be greater than σ distance away from μ or conversely 68.3% of the data will lie within σ distance from μ. Less than 1 in 20 will be greater than 2σ away from μ or conversely 95.5% of the data will lie within 2σ distance from μ. Less than 1 in 300 will be greater than 3σ away from μ or conversely 99.7% of the data will lie within 3σ distance from μ (Adams, 1995; Burns, 2001).

The standard deviation within a data set permits the comparison of the individual data points. A tighter standard deviation (smaller) therefore signifies a more uniform set of data.

2.2.2 Multivariate Analysis

Where section 2.2.1 explained the variance of a single component, there is an increasing emphasis currently being placed on analyzing multi-component (element) samples and utilizing multiple measures in data analysis (Adams, 1995).

To begin understanding the concepts of multi-component analysis, it is important to outline the basic nomenclature used to describe the data. In general, multi-component data is referenced in matrix form to facilitate calculation and organization of the data. Traditionally, data is organized as illustrated in Figure 2-3 (Burns, 2001).

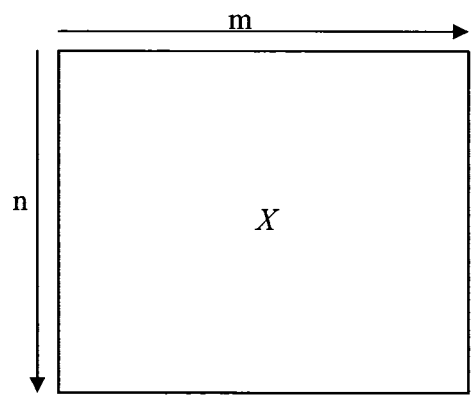

Figure 2-3: Multivariate Array

where
X is the data matrix
n is the number of objects or samples
m is the number of variables or components measured
x_{ij} are elements of the data matrix X

A data matrix with only one row is termed a row vector or "r", and a data matrix with only one column is termed a column vector or "c".

2.2.2.1 Covariance

In a multivariate system it is customary not to analyze the individual variates in isolation, but to combine them in order to provide as complete a description for the system as possible. Variables that display no interaction with any other variables in the system are referred to as *statistically independent*; a change in value of one variable would have no effect on another measured variable (Adams, 1995).

In many cases the variates are not statistically independent. A measure of interaction between variates is required to begin to interpret the data and characterize the samples. The degree of interaction between variables can be estimated by calculating their *covariances* (Burns, 2001).

As variance describes the spread of data about the mean for a single variable, covariance describes the relationship between two variables. The covariance formula in Equation 2-3 is derived from the variance formula in Equation 2-2 (Adams, 1995).

Equation 2-3

$$SP_{kl} = \sum_{i=1}^{n} (x_{ik} - \bar{x}_k)(x_{il} - \bar{x}_l)$$

$$Cov_{kl} = \frac{SP_{kl}}{(n-1)}$$

where
x_{ij} is the i^{th} concentration of variate j.
k, l are two arbitrary variates to be compared.
SP_{kl} is the corrected (mean centered) sum of products of variates k & l.
Cov_{kl} is the covariance coefficient between variates k & l.

2.2.2.2 Correlation

To estimate the degree of interaction between variables, free from the influence of measurement units as in the covariance Equation 2-3, the *correlation coefficient* is introduced. The correlation coefficient requires that the variance of each variate be calculated as in Equation 2-2 (Burns, 2001; Adams, 1995). The correlation coefficient in Equation 2-4 cannot exceed the bounds of +1 to -1, and is therefore normalized to unity.

Equation 2-4

$$s_j^{\,2} = \frac{\sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)}{(n-1)}$$

$$r_{kl} = \frac{Cov_{kl}}{s_k \bullet s_l}$$

where
s_j is the standard deviation of variate j.
r_{kl} is the correlation coefficient between variant k & l.

2.3 Spectroscopic Data Acquisition

2.3.1 Introduction

Davis et al. (2001) describe spectrometry as "the detection and measurement of radiation and its analysis in terms of frequency and energy distribution". Electromagnetic waves carry information about the sources that generate them. Each time-varying wave is composed of a frequency, amplitude and phase. Figure 2-4 depicts two waves with amplitude of one, a frequency of one wave every 2π seconds and a phase difference of $\pi/2$.

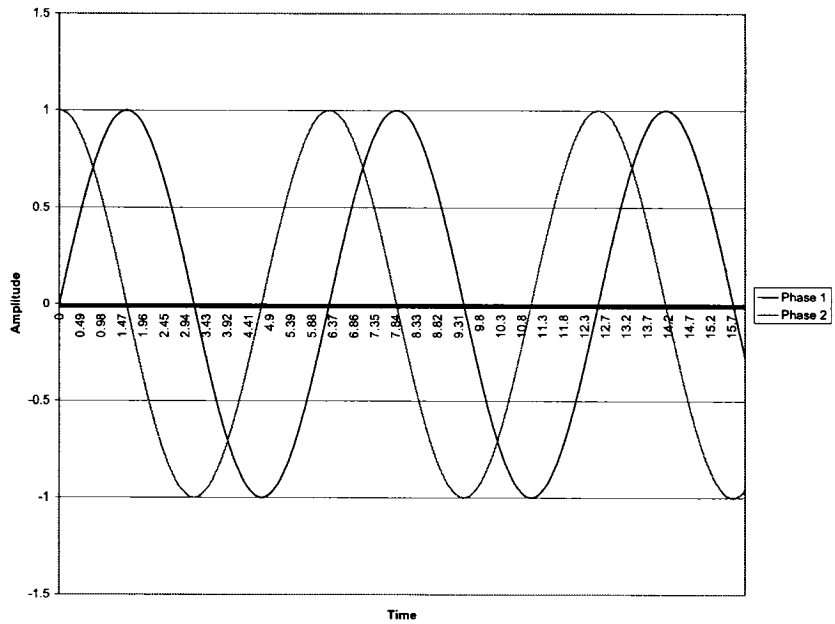

Figure 2-4: Electromagnetic Waves

The majority of electromagnetic sources do not just generate a uniform wave. Sources such as the sun generate many waves with an infinite number of frequencies and amplitudes. Figure 2-5 portrays an example of three waves with different frequencies and amplitudes. The overlay of the three waves is the sum representing the wave that would be measured by a wave-detecting device such as a spectrometer. In the case of the sun, each frequency represents a different color of visible light; when observing the sun, the observer sees only the sum of all the waves that resembles a soft yellow.

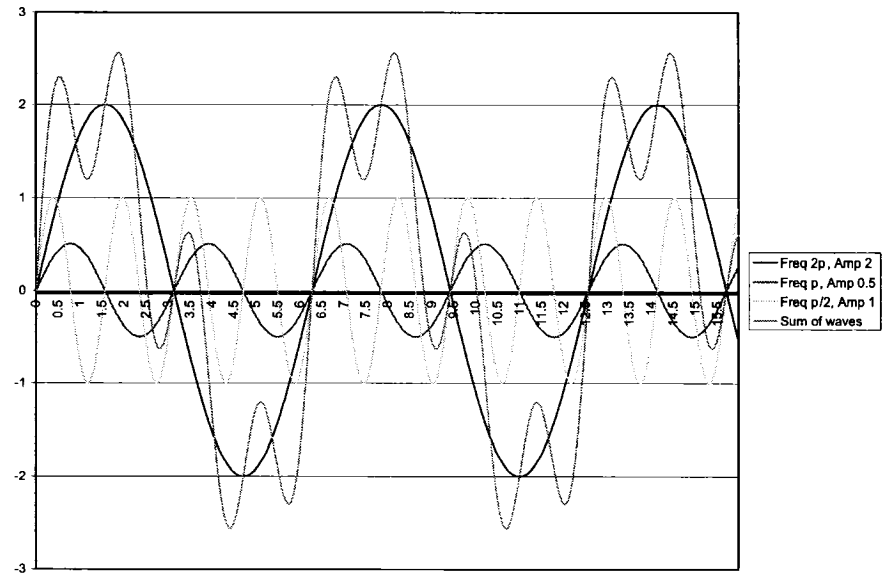

Figure 2-5: Electromagnetic Wave Interference

2.3.2 Fourier Transform

A dispersing element such as a prism may be employed to separate light into its composite frequencies at different angles, producing a "rainbow" of multicolored bands in the case of visible light. Alternatively, the decomposition of a time-varying wave into its individual frequencies may be achieved without the use of a dispersing element through the application of a mathematical algorithm commonly referred to as the *Fourier transform*, (Davis et al., 2001). Applying the Fourier transform to the "sum of waves" plot in Figure 2-5 decomposes the amplitude versus time wave into three waves with frequencies of 2π, π and $\pi/2$ and amplitudes of 2, 0.5 and 1, respectively, as illustrated in Figure 2-6 (Davis et al., 2001).

For the broad-band radiation sources employed in molecular spectroscopy, the development of the Fast Fourier Transform algorithm and high-speed computers were

essential for high-resolution Fourier transform spectroscopy because of the sheer volume of the data and the complexity of the observed waves (Davis et al., 2001).

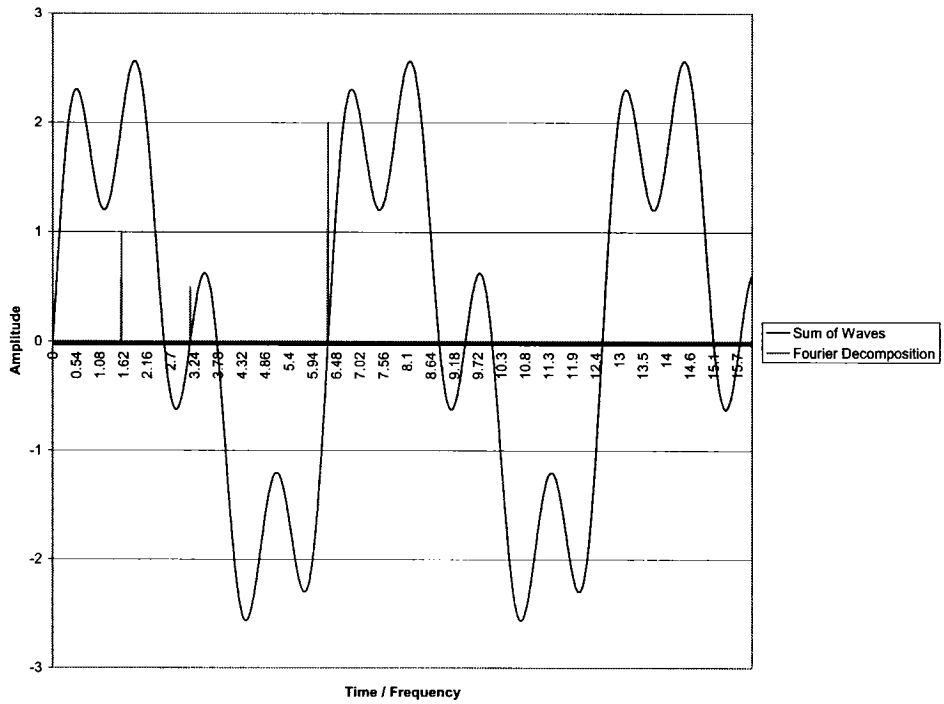

Figure 2-6: Fourier Decomposition

The algorithm for the Fast Fourier Transform or Fourier decomposition of a wave is explained as follows (Bourke, 1993). Consider the wave to be transformed as a vector series of data as illustrated in Equation 2-5.

Equation 2-5

$$x(k) = [x_0, x_1, \ldots, x_{N-1}]$$

where
N is the total number of data points.
x_i is a complex number defined as $x_i = x_{real} + jx_{imaginary}$.
j is the imaginary number, $j = \sqrt{-1}$.

The Fast Fourier Transform of the vector series described by Equation 2-5 will also have N data points and is denoted $X(k)$, and is described by Equation 2-6.

Equation 2-6

$$X(n) = \frac{1}{N}\sum_{k=0}^{N-1} x(k)\exp\left[-\frac{jk2\pi n}{N}\right], \text{ for } n = 0\ldots N\text{-}1$$

2.3.3 Infrared Spectra

Electromagnetic radiation is decomposed into various regions with corresponding wavelengths. In the case of mid-infrared (IR) spectroscopy, λ is in the range of 2.5 to 25 μm. The unit of wavenumber as opposed to wavelength is often used in IR spectroscopy. It is described as the number of waves per centimeter and conforms to the relationship in Equation 2-7 (Davis et al., 2001). Typically, IR spectroscopy covers a wavenumber range of 4000 to 400 cm^{-1}.

Equation 2-7

$$\sigma = \frac{\nu}{c} = \frac{1}{\lambda}$$

where
σ is described as the wavenumber and has the unit cm^{-1}.

2.3.4 Fourier Transform Spectroscopy

To measure electromagnetic waves, most Fourier transform spectrometers make use of a scanning Michelson interferometer. The interferometer records the interferogram of the electromagnetic radiation under examination. The interferogram is then broken down into its frequency components by applying the Fourier transform as illustrated in Equation 2-6 (Davis et al., 2001).

Today's interferometers are recognized for their high optical efficiency, no diffraction losses, high throughput, simultaneous observation of all frequencies / wavelengths, and wide spectral coverage (Davis et al., 2001). In order to comprehend

spectroscopic analysis, it is essential to understand the fundamental theory behind the interferometer.

The process of Fourier transform infrared spectroscopy begins with an infrared source as illustrated in Figure 2-7. The infrared beam is directed toward an interferometer that contains a beam splitter, a fixed mirror, a moving mirror and an optical lens. The beam splitter divides the input source into two beams of equal amplitude. One beam is directed to the fixed mirror and the other to the moving mirror. The two beams are then recombined on the optical lens (often the same beam splitter serves as an efficient lens). The recombined beam is a combination of the two source beams that are allowed to interfere with each other as a function of the moving mirror's displacement.

The output of the interferometer is referred to as an interferogram. The relative intensity of the interferogram is a function of the path difference, X (Figure 2-7), of the moving mirror. The intensity of the interferogram energy is measured in units of Watts per wavenumber or W/cm^{-1}.

When measuring a sample, the interferogram is passed through the sample being analyzed. Depending on the properties of the sample, a portion of the optical energy is absorbed and a portion is transmitted.

The detector is positioned to trap the energy transmitted or reflected, depending on the spectrometer configuration, and outputs the instantaneous interferogram as an analog signal. The interferograms collected by the detector are subjected to a Fast Fourier Transform, which produces a spectrum of energy intensity versus wavelength or wavenumber, referred to as the single beam spectrum. When the single beam spectrum is referenced against another single beam spectrum recorded at ambient conditions with no

sample present (referred to as the 'background'), the absorbance or transmittance spectrum of the sample can be calculated.

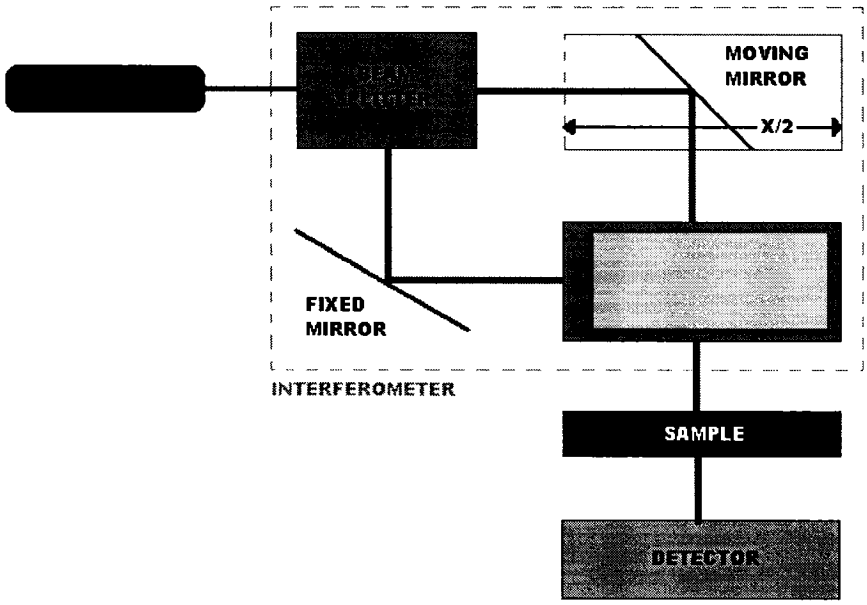

Figure 2-7: The Michelson Spectrometer

The replacement of the detector shown in Figure 2-7 by an array detector provides an "infrared image" of the sample. A focal-plane array (FPA) detector works similarly to a digital camera. As opposed to collecting a single spectrum at a time, imaging spectrometers collect hundreds if not thousands of spectra simultaneously. Each of the spectra corresponds to the signal recorded at a single pixel on the FPA. Accordingly, the spatial resolution of an imaging spectrometer is defined by the field of view divided by the square of the number of pixels in the array.

2.3.5 Spectral Resolution

The cleanness of an apparatus function and the precision of the wavenumber and intensity scales and any possible sources of excess noise must be determined to produce a reliable data set (Davis et al., 2001). The maximum path difference, X, between the interfering beams in a Fourier transform spectrometer (see Figure 2-7) dictates the resolution of the instrument. The instrument resolution is determined by taking the inverse of the maximum path difference as in Equation 2-8.

Equation 2-8

$$\text{Resolution} = \frac{1}{X_{MAX}}$$

For example, if the maximum path difference of an instrument is five meters (five hundred centimeters), then the corresponding resolution will be 0.02 cm^{-1}. Also the resolution of the instrument can be interchangeably used as the absolute wavenumber precision of the instrument. Instruments with variable resolution must be utilized carefully; excessive resolution deteriorates the quality of the signal to noise ratio.

2.3.6 Signal and Noise

Even under the strictest of experimental conditions, there are always various kinds of noise generated by the source, electrical and mechanical variations in the environment or in the spectrometer itself (Davis et al., 2001). Noise is identifiable by sharp spikes, false spectral lines or other features not predictable by the properties of the incoming radiation.

Physical noise is generated when the interferogram is collected and therefore it is important to understand the physical aspects when measuring a sample. Varying

procedures, processes and events at the instrument will translate into varying noise levels in the final spectrum (Davis et al., 2001).

The signal to noise ratio is a method of calculating the strength of the signal vs. the ambient noise as illustrated in Equation 2-9 (Adams, 1995).

Equation 2-9

$$S/N = \frac{average\_signal\_magnitude}{RMS\_Noise}$$

$$RMS\_Noise = \sqrt{\frac{\sum_{i=1}^{n}(\bar{x} - x_i)^2}{n-1}}$$

where
n is the number of samples present
x_i is the signal
$\bar{x}$ is the average signal

RMS noise can be identified as the standard deviation (σ) of the noise signal; therefore, the signal to noise ratio from Equation 2-9 can be redefined by Equation 2-10 (Adams, 1995).

Equation 2-10

$$S/N = \frac{\bar{x}}{\sigma}$$

where
σ is the standard deviation of the noise signal.

2.4 Data Enhancement

2.4.1 Reducing Noise

Numerous spectral anomalies including instrumental noise, random and natural variation in a sample's characteristics and composition as well as atmospheric conditions make an exact match between two spectra of the same substance almost impossible to

obtain; random errors will always exist. This section will explore methods of reducing noise.

2.4.1.1 Co-Adding Spectra

Signal averaging is a process that is conducted by co-adding individual spectra (Adams, 1995). Assuming that noise is randomly distributed, signals are enhanced because the signal strength or magnitude grows linearly with the number of scans N.

Equation 2-11

$$\text{Signal Magnitude} \propto N = k_1 N$$

In a similar fashion, the effects of the variances of noise grow linearly with each successive scan. RMS noise is equated with the standard deviation (being the square root of the variance) and therefore the magnitude of noise can be expressed as a function of the square root of the number of scans. As a result, the signal to noise ratio is increased linearly in proportion to the square root of the number of scans taken as in Equation 2-12.

Equation 2-12

$$\frac{Signal}{Noise} = \frac{k_1 N}{k_2 \sqrt{N}} = k\sqrt{N}$$

A common practice in infrared spectroscopy is to co-add 100 spectra in order to receive a 10:1 theoretical enhancement in signal to noise ratio. Co-adding the spectra can be implemented in the collection stage, or after all the samples have been scanned once each. When using a focal plane array detector, the signals recorded by individual pixels can be co-added to reduce the effect of infrared refraction from pixel to pixel as well as the overall noise due to imperfections in the super-conductor (Wolsky et al., 1989).

2.4.1.2 Smoothing

There are a wide variety of signal-smoothing algorithms available for smoothing spectral data. The most fundamental of these methods is referred to as the *boxcar method* or a *mean smoother* (Adams, 1995; Beebe et al., 1998).

The boxcar method involves dividing a spectrum into equally spaced segments of five, seven, nine or eleven points. The centroid of each of these data segments is calculated. The data points in the segment analyzed are then replaced with the calculated value at the centroid location.

The centroid location is determined by calculating the center of mass of the data points. The interpretation of a value for mass is questionable. Some analysts will use a consistent value of 1 as the mass at any given equally spaced data point. Others will utilize a formula proportional to the absorbance at the particular data point. Either way, the general X, Y coordinates of a centroid are calculated as in Equation 2-13.

Equation 2-13

$$\overline{X} = \frac{\sum x_i m_i}{\sum m_i} \quad \overline{Y} = \frac{\sum y_i m_i}{\sum m_i}$$

where

$\overline{X}$ is the x-axis (wavelength) location of the centroid.
$\overline{Y}$ is the y-axis (absorbance) location of the centroid.
x_i is the given x-axis cordinate of a particular data point *i*.
y_i is the given y-axis coordinate (absorbance) of a particular data point *i*.
m_i is the given mass of a particular data point *i*.

Although the boxcar method is an excellent method of smoothing a spectrum, it increases the distortion of the signal. Subsequently, boxcar smoothing results in a loss of spectral resolution due to the fewer data points available for analysis (Adams, 1995).

The *moving average method* of smoothing is similar to the boxcar method but provides a more stable method of maintaining spectral resolution. As opposed to replacing a series of data points with one centroid data point, the moving average method replaces each data point with a value averaged from those points surrounding it (Beebe et al., 1998). For example, if a five point moving average, or *running mean smoother*, is performed, data point x_3 is replaced with the average value of data points x_1, x_2, x_3, x_4 and x_5. In turn, data point x_4 is replaced with an average value of data points x_2, x_3, x_4, x_5 and x_6 (Beebe et al., 1998).

The mathematical process of implementing a moving average is referred to as a *convolution*. As with the boxcar method, convolution is a function of mass. However, when applying a moving average, the mass at each point is equivalent to unity, or 1. The formula for applying the convolution is expressed in Equation 2-14.

Equation 2-14

$$x_i' = \frac{\sum_{j=-n}^{n} x_{i+j} m_j}{\sum_{j=-n}^{n} m_j} = \frac{\sum_{j=-n}^{n} x_{i+j}}{(2n+1)}$$

where
n is the incremental number of points to average in each direction.

Polynomial smoothing extends the concept of a moving average by modifying the mass vector, such that the mass vector now describes a convex polynomial (Adams, 1995). Without going into detail, it turns out that replacing values of m_j with specific predetermined constants and performing a moving average is equivalent to calculating a polynomial function for each increment of data points (n) (Beebe et al., 1998). These

constants are referred to as Savitzky-Golay coefficients after their founders and are listed in Table 2-1.

Table 2-1: Savitzky-Golay Coefficients

i/n	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8
5							-3	12	17	12	-3						
7						-2	3	6	7	6	3	-2					
9					-21	14	39	54	59	54	39	14	-21				
11				-36	9	44	69	84	89	84	69	44	9	-36			
13			-11	0	9	16	21	24	25	24	21	16	9	0	-11		
17	-21	-6	7	18	27	34	39	42	43	42	39	34	27	18	7	-6	-21

For example, when applying a five point ($n = 2$) moving average, the mass vector is replaced with the following coefficients:

$$m_{-2} = -3$$
$$m_{-1} = 12$$
$$m_0 = 17$$
$$m_1 = 12$$
$$m_2 = -3$$

One of the disadvantages of running smoothing functions is the so-called "end effects." When the smoothing function is running on the first few or last few data points, not enough data points are available to completely smooth the sample (Beebe et al., 1998). Care must be taken not to put too much emphasis on the end data points during analysis.

2.5 Feature Selection and Extraction

2.5.1 Introduction

Post data collection and data enhancement and prior to analyzing the data by calibration, modeling or pattern recognition techniques, it is usual to perform some pre-

processing of the data. Typically, there are three principal aims in the pre-processing of the data collected (Adams, 1995).

1. To reduce the amount of data and eliminate data that is irrelevant to the task being undertaken.
2. To preserve or enhance sufficient information within the data in order to achieve certain goals.
3. To extract the information in a form suitable for further analysis.

The techniques used to pre-process the data are referred to as *feature selection* and *feature extraction*. Feature selection is defined as identifying and selecting features in analytical data that are believed to be important in calibration or pattern recognition. Feature extraction changes the dimensionality of the data and generally refers to combining or transforming original variables to provide better new ones. This section defines selected commonly implemented methods of feature selection and feature extraction.

2.5.2 Feature Selection

Various data manipulation techniques are commonly employed to assist in feature selection. These techniques include means of accentuating the spectral differences within a data set and compensating for extraneous sources of spectral variation irrelevant to the study being undertaken (Adams, 1995).

2.5.2.1 Mean Centering

Mean centering is a process used when dealing with a large number of samples. Essentially, the mean of each variable is subtracted from each of the samples (Beebe et al., 1998). Data sets are often mean centered to account for intercepts in regression models (Beebe et al., 1998). Mean centering generally does not hurt data, and often helps; therefore, many analysts mean center their data as a default. Mean centering is always recommended when performing routines such as Principal Component Analysis (see Section 2.5.3).

2.5.2.2 Normalization

Normalization is perhaps the most common form of data pre-processing used today (Adams, 1995). In its simplest terms, normalization involves scaling spectral data to a given constant. Normalization is used to remove systematic variation usually associated with the total amount or thickness of the sample under investigation (Beebe et al., 1998).

One method of normalization, referred to as normalizing to unit intensity, involves identifying the absolute highest peak in a spectrum. Subsequently, all the data points in the spectrum are divided by the absolute value of the largest peak height (Beebe et al., 1998).

Normalizing to unit area. termed *1-norm* normalization, is accomplished by calculating a 1-norm constant as shown in

Equation 2-15, and dividing every point in the spectrum by that constant (Beebe et al., 1998).

Equation 2-15

$$1-norm=\sum_{j=1}^{n}\left|x_j\right|$$

where
n is the number of data points in the spectrum.

Similarly, normalizing to unit length, termed *2-norm* normalization, is accomplished by calculating a 2-norm constant as shown in Equation 2-16, and dividing every point in the spectrum by that constant (Beebe et al., 1998).

Equation 2-16

$$2-norm=\sqrt{\sum_{j=1}^{n}x_j^2}$$

2.5.2.3 Baseline Correction

Aside from noise, measured signals can also contain low-frequency variations not related to the sample under study. These components are referred to as baseline features and can be relatively large if not removed.

The theory of baseline correction is that any sample vector can be written as a function of x as in Equation 2-17. The function is equal to the sum of the actual signal plus some baseline feature that can be expressed in polynomial form (Beebe et al., 1998).

Equation 2-17

$$r=f(x)=\tilde{r}+\alpha+\beta x+\gamma x^2+\delta x^3+\ldots$$

where
$\tilde{r}$ is the signal of interest.
$\alpha+\beta x+\gamma x^2+\delta x^3+\ldots$ is a polynomial approximating the baseline feature.

By postulating an algebraic model for the baseline as offset, linear or polynomial, the baseline component of the signal can be accounted for by simple subtraction.

An offset baseline correction (i.e. a horizontal line) can be expressed as Equation 2-18. The baseline can be removed by estimating a value for α and subtracting it from every element in the vector *r*. The optimal value for α would be found by selecting a point on the original vector that is known to contain only background or baseline information. The average intensity of several baseline variables is often used in order to eliminate the amount of noise introduced into the sample vector by baseline subtraction.

Equation 2-18

$$r = \tilde{r} + \alpha$$

A linearly sloping baseline is quite common in spectroscopy due to wavelength dependent scattering. Linear baseline correction is expressed as in Equation 2-19. In this case, a line is estimated and two or more points assumed to contain only baseline information are required to solve for the baseline constants, α and β.

Equation 2-19

$$r = \tilde{r} + \alpha + \beta x$$

Other functions can also be estimated as long as the reference points selected are only influenced by the baseline. If reference points are chosen poorly, chemical variation in the data will be removed in addition to the baseline (Beebe et al., 1998).

2.5.2.4 Derivatives

Derivative spectroscopy provides another method of eliminating the baseline features from a spectrum (Beebe et al., 1998). It also provides a window to analyze data in a potentially more useful form than the zero'th order (Adams, 1995).

Referring to the baseline equation (Equation 2-17) and taking the first derivative of the sample vector with respect to the variable x yields Equation 2-20 (Beebe et al., 1998).

Equation 2-20

$$\frac{dr}{dx} = r' = \tilde{r}' + 0 + \beta + 2\gamma x + 3\delta x^2 + \ldots$$

Equation 2-20 reveals that the first derivative has completely removed the offset feature, α. If the baseline is only comprised of an offset, the other coefficients in Equation 2-20 would be zero as well, and the baseline effect eliminated.

If a more complex baseline exists, then each successive derivative will successfully remove a higher order term as illustrated in Equation 2-21.

Equation 2-21

$$\frac{d^2r}{dx^2} = r'' = \tilde{r}'' + 0 + 0 + 2\gamma + 6\delta x + \ldots$$

As well as eliminating the baseline component of the samples, analytical applications of derivative spectroscopy are numerous and are usually a result of the higher resolution of the differential data with respect to the original data. Derivative spectroscopy enhances changes in slope that are typically difficult to extract from their zero'th order counterparts. The downfall of derivative spectroscopy is that it also greatly increases the effect of noise apparent in the original data. Hence, this limits derivative analysis to spectra with a high signal to noise ratio and emphasizes the importance of enhancing the spectral data prior to pre-processing (Adams, 1995).

Several mathematical algorithms exist for differentiating spectral data. All of these methods require the data points to be evenly spaced (in most cases, a given wavenumber or wavelength interval). In cases when constant data intervals cannot be

recorded, techniques such as interpolation must be employed to extract the necessary data.

The simplest method of computing the first-order derivative is calculated by Equation 2-22.

Equation 2-22

$$\frac{dy}{d\lambda} = \frac{y_{i+1} - y_{i-1}}{2\Delta\lambda}$$

where
$\Delta\lambda$ is the given measurement interval (resolution) of the data.
y_i is the given response (i.e. absorbance or intensity) at a data point *i*.

Similarly, the second derivative is calculated by Equation 2-23.

Equation 2-23

$$\frac{d^2y}{d\lambda^2} = \frac{y_{i+1} - 2y_i - y_{i-1}}{\Delta\lambda^2}$$

Various other methods exist for computing the first and second derivative. For example, using the Savitzky and Golay smoothing techniques outlined in Section 2.4.1.2, the effect of noise can be significantly reduced with their weighing techniques and a better approximation can be formed (Adams, 1995). The Savitzky and Golay first-order and second-order derivatives are described by Equation 2-25

Savitzky and Golay propose the first-order derivative as described by Equations 2-24 and 2-25, respectively.

Equation 2-24

$$\frac{dy}{d\lambda} = \frac{1}{10\Delta\lambda}\left(-2y_{i-2} - y_{i-1} + y_{i+1} + 2y_{i+2}\right)$$

Equation 2-25

$$\frac{d^2y}{d\lambda^2} = \frac{1}{7\Delta\lambda^2}\left(2y_{i-2} - y_{i-1} - 2y_i - y_{i+1} + 2y_{i+2}\right)$$

2.5.2.5 Integration

Integration is naturally the complement to differentiation in mathematical terms. In its elementary definition, the integral is simply the area under the curve of the spectrum. Many methods of computing the area under a curve exist. In principle, they involve dividing the curve into rectangles or trapezoids in order to estimate the area under the curve. As the number of data points increases within a given spectrum, so does the accuracy of such computational algorithms. Although both the rectangular and the trapezoidal integration method would be applicable to obtain a crude estimate of the integral, there is a more reliable and better-suited method derived from combining both the rectangular and trapezoidal algorithms. This method is referred to as *Simpsons Method* and is described by Equation 2-26.

Equation 2-26

$$A_{i+0.5} = (x_{i+1} - x_i)\frac{(4y_{i+0.5} + y_{i+1} + y_i)}{6}$$

2.5.3 Feature Extraction

In the interest of increased computational efficiency and improved analytical differentiation, data can be combined linearly to produce new variables. In essence, a linear combination of variables is represented by replacing two or more correlated variables with a weighted sum of those variables. This new variable sits on a new axis at some arbitrary angle α from the original axis as outlined in Equation 2-27 (Adams, 1995).

Equation 2-27

$$X = a \bullet X1 + b \bullet X2$$
$$a^2 + b^2 = 1$$
$$a = \sin\alpha$$
$$b = \cos\alpha$$

where
$X1$ and $X2$ are the original variables.
X is the new variable
a and b are the normalized weights.
α is the angle of the new axis.
If $a = b = 1/\sqrt{2}$ then $\alpha = 45°$.

If there is correlation between $X1$ and $X2$, then the variance of X will be significantly greater than the individual variances of $X1$ and $X2$, respectively. Therefore, the new variable X contains more useful information than either of the variables $X1$ and $X2$.

When dealing with more than two variables, the variance of the new variable can be described using the covariance or correlation coefficient (as discussed in Section 2.2.2) as described by Equation 2-28.

Equation 2-28

$$X = a_1 x_1 + a_2 x_2 + ... + a_n x_n$$
$$s_X{}^2 = \sum_{j=1}^{n}\sum_{k=1}^{n} a_j \bullet a_k \bullet Cov_{jk}$$
$$s_X{}^2 = \sum_{j=1}^{n} a_j{}^2 \bullet s_j{}^2 + \sum_{j=1}^{n}\sum_{k=j+1}^{n} a_j \bullet s_j \bullet a_k \bullet s_k \bullet r_{jk}$$

where
r_{jk} is the correlation coefficient between j and k.

2.5.3.1 Principal Component Analysis

Principal Component Analysis (PCA) involves the rotation and transformation of the original n axes, each representing a variable, to a new set of axes. These new axes

provide the maximum level of variance between the variables and ensure that they are orthogonal (perpendicular) and uncorrelated. Since PCA usually produces a new set of variables p, where p is always less than n, PCA proves to be a useful technique in reducing the dimensionality of the sample data (Adams, 1995; Beebe et al., 1998).

PCA functions as an algorithm to seek out the first principal component, or principal axis that is able to inclusively reflect the greatest amount of variance in the data. Once the first principal component is identified, the search continues to find the second principal component. The second principal component is able to inclusively reflect the greatest amount of variance in the remaining data and is completely uncorrelated with the first principal component. The algorithm repeats until all of the principal components have been identified and accounted for (Beebe et al., 1998).

In essence, if two variables with a certain degree of covariance $VAR1 \& VAR2$ exist, then as discussed in Section 2.2.2.1, the covariance $Cov_{VAR1,VAR2}$ can be determined. As well, the variance of each of the variables s^2_{VAR1} & s^2_{VAR2} is known. If the two variances are plotted on the x and y axes of a Cartesian plane respectively, at a distance equivalent to the covariance, perpendicular to their axes, then it is easy to visualize the first principal component. The first principal component is the axis drawn through the center of the ellipse formed by the origin of the plane (as the center) and the two data points as illustrated in Figure 2-8 (Adams, 1995)

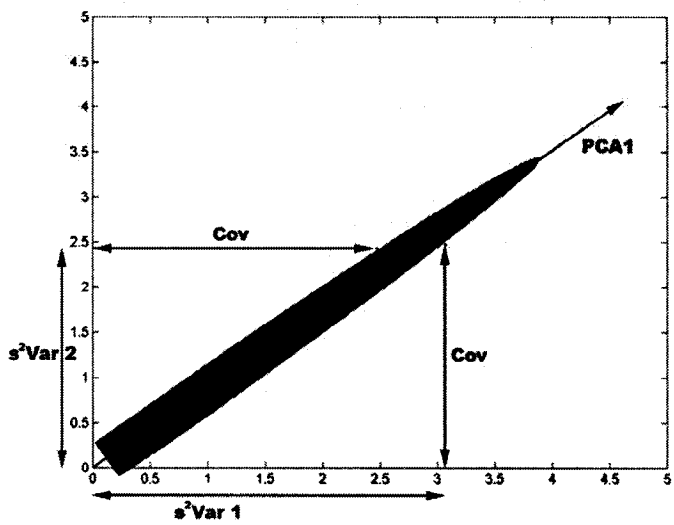

Figure 2-8: First PCA Axis

Mathematically, the first principal component slope is equal to the *eigenvector* of the variance and covariance matrix, and the length of the axis is equal to the calculated *eigenvalue* for the eigenvector. In the same manner, the second principal component's slope is equal to the second eigenvector and its length to the second eigenvalue (Adams, 1995).

Principal component analysis is employed extensively in infrared spectroscopy. The principal component loadings or eigenvectors highlight the weights given to each spectral point in each of the original spectra. The results of principal component analysis reduce dimensionality, and therefore, the similarity and differences between samples can often be better assessed. Consequently, principal component analysis is a cornerstone in chemometric analysis (Beebe et al., 1998).

2.6 Pattern Recognition

2.6.1 Introduction

Classification arises from the need to highlight similarities and differences between samples collected as modern analytical techniques generate large amounts of both qualitative and quantitative data (Adams, 1995). The purpose of classification is to derive a mathematical scheme for grouping into classes such that objects within a class are similar and different from those in other classes.

Supervised pattern recognition or *supervised learning* requires a training set where the parent class group of each sample is known. This information can be used to develop functions suitable for classifying unknown samples.

Unsupervised pattern recognition or *cluster analysis* consists of classifying a group of data where no class is known or identified.

2.6.2 Measuring Distances between Objects

Pattern recognition procedures typically begin with the calculation of a matrix of similarities or dissimilarities between the objects. Similarity and distance between objects are complementary concepts with no formal definition. In practice, distance as a measure of dissimilarity is a much more clearly defined quantity and is therefore more commonly used in pattern recognition (Adams, 1995).

The first stage of any pattern recognition procedure relies on the proper selection of a distance measure. It is recommended that clustering techniques be repeated with different distance measures in order to determine the proper fit for the data at hand.

In most applications of cluster analysis, the correlation coefficient used in similarity measures is too limiting (Adams, 1995). Correlation coefficients are solely a measure of colinearity between variates and do not take into account non-linear relationships, or the absolute magnitude of the variates under analysis. Distance measures are more commonly encountered in cluster analysis because of their accurate representation of the variates and their ability to be represented mathematically. However, it is always possible at the end of a cluster analysis to represent the data as a reverse similarity; the greater the distance between objects, the less their similarity.

Any object is characterized by a set of measures and can therefore be represented as a point in multivariate space defined by axes. Each axis corresponds to a variate that describes the object. For example, consider two objects A and B each described by two variates on a Cartesian coordinate system. Object A is characterized by vector $a = x_{11}, x_{12}$, and object B is characterized by vector $b = x_{21}, x_{22}$.

When using a distance measure, the objects closest together are assigned to the same cluster. For a distance function to be useful, the following rules must apply (for objects A and B only).

(a) $d_{AB} \geq 0$, the distance between all pairs of measurements for object 'A' and object 'B' must be non-negative.

(b) $d_{AB} = d_{BA}$, the distance measure is symmetric and can only be zero when $A = B$.

(c) $d_{AC} + d_{BC} \geq d_{AB}$, the distance is commutative for all pairs of points.

The most common distance measure is referred to as the Minkowski measure and is described by Equation 2-29.

Equation 2-29

$$d_{AB} = \left[\sum_j \left(x_{1_j} - x_{2_j} \right)^m \right]^{1/m}$$

where
x_{i_j} is the value of the j^{th} variable measured on the i^{th} object.
m is a constant related to the metric used.

If $m = 1$, than the equation is referred to as the *city-block metric*. The most common distance measure used is when $m = 2$ and is described as the *Euclidean distance* (Beebe et al., 1998).

Figure 2-9 illustrates the difference between the city-block and Euclidean distance measures (Adams, 1995).

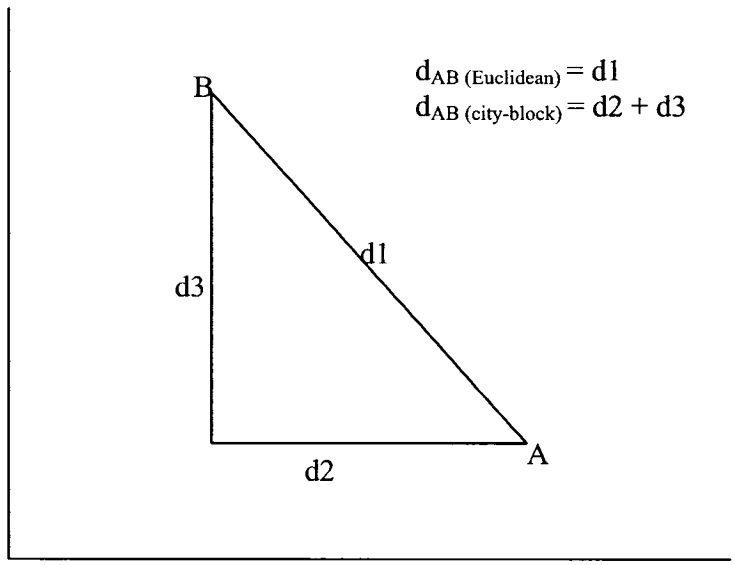

Figure 2-9: Distance Measures

A suitable distance measure for pattern recognition can now be examined. A simple example serves to illustrate the principal points. Table 2-2 describes three objects ('A', 'B' and 'C') each characterized by five variates.

Table 2-2: Distance Measure Example – Sample Data

	X_1	X_2	X_3	X_4	X_5
A	2.1	5.2	3.1	4.1	2.1
B	2.5	4.0	4.0	4.6	3.5
C	5.1	9.2	7.1	7.0	5.0

Table 2-3 illustrates the tabulated results of using the Euclidean distance measure; in this case the smallest distances are presented in bold face. Figure 2-10 illustrates the corresponding dendrogram.

Table 2-3: Distance Measure Example – Distances

	A	B	C
A	0	**2.15**	7.60
B	**2.15**	0	**7.17**
C	7.60	7.17	0

	AB	C
AC	0	**7.38**
B	**7.38**	0

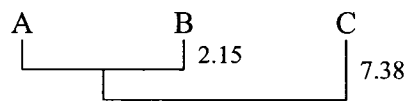

Figure 2-10: Distance Measure Example – Dendrogram

It is apparent that different results are obtained when using different measures. Figure 2-11 illustrates the objects from the sample data set in Table 2-2 plotted with their variables, and an explanation of the different results is evident. If the variables in the sample represent trace elements in a water sample for instance, then samples 'A' and 'B' are similar with the subtle difference possibly due to experimental error. Sample 'C' would be from a different source as its elemental concentrations are significantly different. In this case, the distance metric would be a suitable clustering measure. On the other hand, if the data represented points in a spectrum, than spectra 'A' and 'C' would be similar while differing only in scale. Spectrum 'B' would have a completely different

profile. In this case the correlation metric would prove to be a suitable method for clustering. If, however, the spectra had been normalized about the most intense response, then spectra 'A' and 'C' would be closer and the distance metric more meaningful.

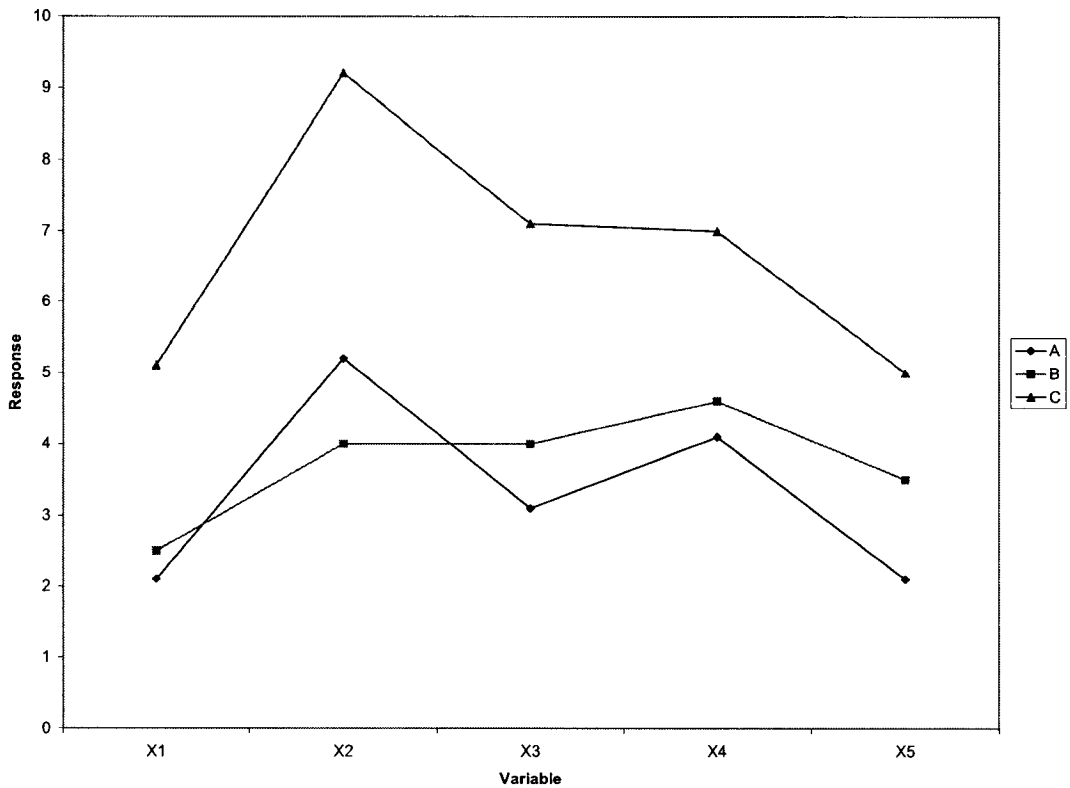

Figure 2-11: Distance Measure Example – Data Plot

2.6.3 Unsupervised Clustering Techniques

Figure 2-12 illustrates a visual example of clustering. It is evident from inspection that there are several ways of dividing the pattern space and producing several different clusters of objects. There is no single correct result; the success of any clustering method is dependent on what is being sought, and the intended use of the clustered information.

When class information is known about the data set, it is of interest to compare this information to the natural classification. The natural clustering might or might not

relate to the expected groupings. If the natural clustering does not match the expected classification, then this indicates a disconnect between the measurements chosen, the data pre-treatment techniques or the data pre-processing techniques, and the expected results (Beebe et al., 1998).

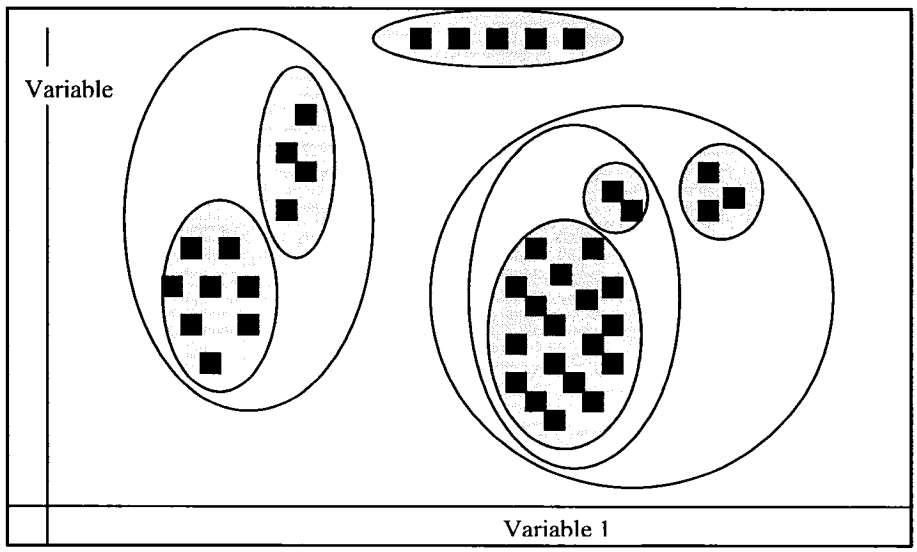

Figure 2-12: 2-D Cluster Example

The general algorithm used for applying unsupervised pattern recognition proceeds in the following manner.

1. The raw pre-processed data characterizing the samples being clustered are converted to a set of similarity and dissimilarity measures between samples.
2. The aim is to cluster the samples with little separation between samples of the same class while maintaining separation between different clusters.

When grouping objects together to form a cluster, the cluster itself can be represented by a typical member of the cluster. A typical member could be an actual

object within the cluster, or more commonly an object constructed of the mean variate values of the objects within the cluster. The between-cluster distance can then be defined by some metric such as the Euclidean distance between these means. The *nearest neighbor* distance describes the distance between the two closest members from different groups. On the other hand, the *furthest neighbor* distance describes the distance between the two furthest members from different groups. Further inter-group measures are obtained by taking the average inter-element measurements between elements in different groups (Adams, 1995).

When only two or three variables are measured for each object, clusters can usually be visually identified. As the number of variates increases, visual interpretation becomes difficult and often clusters are missed. To address this problem, clustering techniques have been developed and are classified as the following types (Adams, 1995).

(a) Hierarchical techniques in which objects are clustered together to form new representative objects. The process is repeated at different levels to produce a dendrogram.

(b) Optimization of the partitioning between clusters using an iterative algorithm until some minimal change in the clustered groups occurs.

(c) Fuzzy cluster analysis in which objects are assigned a membership function indicating their degree of belonging to a certain cluster.

2.6.3.1 K-Means Clustering

In order to explain this clustering method, a set of sample data from Adams (1995) will be utilized. Table 2-4 illustrates 12 samples or objects with two variables

each. Table 2-5 illustrates the corresponding Euclidean distance matrix for this data. Figure 2-13 illustrates the relationship between the samples based on the two variables. Preliminary examination of the data graphed in Figure 2-13 reveals a single outlier point (L) and three distinct groups of data (B,C,D), (A,E,F,G) and (H,I,J,K).

Table 2-4: K-Means Example – Sample Data

	A	B	C	D	E	F	G	H	I	J	K	L
X1	2	6	7	8	1	3	2	7	6	7	6	2
X2	1	1	1	1	2	2	3	3	4	4	5	6

Table 2-5: K-Means Example – Euclidean Distance Matrix

	A	B	C	D	E	F	G	H	I	J	K	L
A	0.0	4.0	5.0	6.0	1.4	1.4	2.0	5.4	5.0	5.8	5.7	5.0
B	4.0	0.0	1.0	2.0	5.1	3.2	4.5	2.2	3.0	3.2	4.0	6.4
C	5.0	1.0	0.0	1.0	6.1	4.1	5.4	2.0	3.2	3.0	4.1	7.1
D	6.0	2.0	1.0	0.0	7.1	5.1	6.3	2.2	3.6	3.2	4.5	7.8
E	1.4	5.1	6.1	7.1	0.0	2.0	1.4	6.1	5.4	6.3	5.8	4.1
F	1.4	3.2	4.1	5.1	2.0	0.0	1.4	4.1	3.6	4.5	4.2	4.1
G	2.0	4.5	5.4	6.3	1.4	1.4	0.0	5.0	4.1	5.1	4.5	3.0
H	5.4	2.2	2.0	2.2	6.1	4.1	5.0	0.0	1.4	1.0	2.2	5.8
I	5.0	3.0	3.2	3.6	5.4	3.6	4.1	1.4	0.0	1.0	1.0	4.5
J	5.8	3.2	3.0	3.2	6.3	4.5	5.1	1.0	1.0	0.0	1.4	5.4
K	5.7	4.0	4.1	4.5	5.8	4.2	4.5	2.2	1.0	1.4	0.0	4.1
L	5.0	6.4	7.1	7.8	4.1	4.1	3.0	5.8	4.5	5.4	4.1	0.0

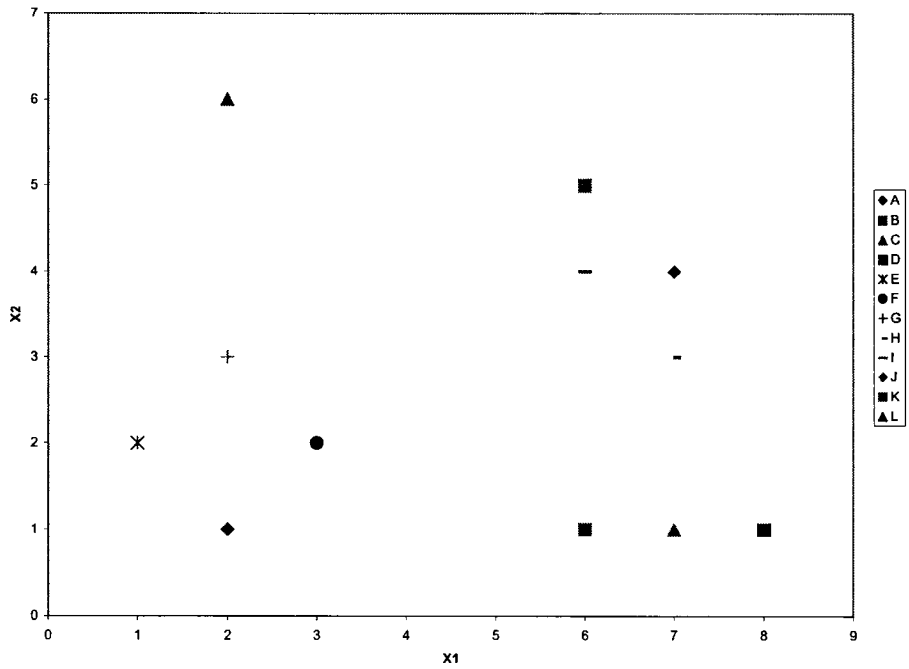

Figure 2-13: K-Means Example – 2-D Variable Plot

The *K-Means* algorithm is one of the most popular and widely used clustering techniques due to its advantage in being applied to relatively large sets of data. K-Means is an optimization-based technique aimed at partitioning m objects, characterized by n variables, into K (user specified) number of clusters (Adams, 1995).

The K-Means method relies on reducing the square of the within-cluster sum of distances. In practice, K-Means cannot be expected to predict the best possible partitioning of the data as it is only a local optimization algorithm. Local optimum in this classification method is obtained when no movement of an object from one cluster to another will reduce the within-cluster sum of squares (Adams, 1995).

Although several K-Means algorithms exist, the *Hartigan* method is the most commonly used (Adams, 1995). The Hartigan algorithm requires that a matrix X be defined with elements $x_{i,j}$, where $(1 \leq i \leq m, 1 \leq j \leq n)$.

L is defined as an arbitrary cluster and the number of objects residing in cluster L *is* denoted R_L; where R_L is defined as the total *responsibility* for the objects residing in L. The mean value of each variable j from all the objects residing in cluster L is denoted $B_{L,j}, (1 \le L \le K)$; where $B_{L,j}$ is defined as the center of the cluster L (MacKay, 2003).

The distance between the i^{th} object and the center of each cluster is given by the Euclidean metric in Equation 2-30. The error (ε) associated with any partition is defined by Equation 2-31 as the sum of the squares of the distance between the i^{th} object and the center of the cluster in which the object resides (MacKay, 2003).

Equation 2-30

$$D_{i,L} = \left[\left(x_{i,j} - B_{L,j}\right)^2\right]^{1/2}$$

Equation 2-31

$$\varepsilon = \sum \left(D_{iL(i)}\right)^2$$

where
$L(i)$ is the cluster containing the i^{th} object.

The K-means algorithm aims to move an object from one cluster to another in order to reduce the error ε and ends when no movement can further reduce ε (Adams, 1995). The algorithm is outlined as follows:

(a) Define a number of clusters, K. Initially assign each of the objects i to one of the clusters. Equation 2-32 is a common method of assigning objects i to clusters $L(i)$.

Equation 2-32

$$L(i) = INT\left[(K-1)\left(\frac{\sum X_{i,j} - MIN\sum X_{i,j}}{MAX\sum_j X_{i,j} - MIN\sum_j X_{i,j}}\right)\right] + 1$$

where

$\sum_j X_{i,j}$ is the sum of all the variables for each object.

MIN and MAX denote the minimum and maximum sum values.

(b) Given a number of predefined clusters, K, and their initial contents, calculate the cluster means $B_{L,j}$ and the initial partition error ε per Equation 2-31.

(c) For the first object, i = 1, compute the increase in error $\Delta\varepsilon$ obtained by transferring the object from the current cluster ($L(1)$) to every other cluster L, $(2 \le L \le K)$ as defined by Equation 2-33. If the $\Delta\varepsilon$ value is negative, the move would reduce the total partition error, and the object should be transferred from the initial cluster to the cluster L. The cluster means, $B_{L,j}$, should be adjusted accordingly to compensate for their new populations.

Equation 2-33

$$\Delta\varepsilon = \frac{(R_L)(D_{1,L})^2}{(R_L)+1} - \frac{(R_{L(1)})(D_{1,L(1)})^2}{(R_{L(1)})-1}$$

(d) Repeat step (c) for every object in the data space.

(e) If no object has been moved, then stop; otherwise return to step (c).

To further illustrate the functionality of the K-Means algorithm, consider the example provided (Adams, 1995). The first step involves specifying the number of clusters (K) expected, in this example K = 4. The next step requires that each object must be assigned to an initial cluster by applying Equation 2-32; the variable sum results are tabulated in Table 2-6.

Table 2-6: K-Means Example – First Iteration Sums

	A	B	C	D	E	F	G	H	I	J	K	L
x_1	2	6	7	8	1	3	2	7	6	7	6	2
x_2	1	1	1	1	2	2	3	3	4	4	5	6
$\sum_j X_{i,j}$	3	7	8	9	3	5	5	10	10	11	11	8

The maximum and minimum variables sums are identified as 11 and 3, respectively, and these values are plugged into Equation 2-32 for object 'A' to produce the results in Equation 2-34.

Equation 2-34

$$L(A) = INT\left\{(4-1)\left[\frac{(3-3)}{(11-3)}\right]\right\}+1 = 1$$

The results for each object are tabulated in Table 2-7. It is apparent that objects (A, E, F, G) are assigned to cluster 1, (B, C, L) are assigned to cluster 2, (D, H, I) to cluster 3 and (J, K) to cluster 4.

Table 2-7: K-Means Example – First Iteration Cluster Assignments

i=	A	B	C	D	E	F	G	H	I	J	K	L
L(*i*)	1	2	2	3	1	1	1	3	3	4	4	2

The next step involves calculating a value for the centers of the clusters. For cluster 1, the centers are calculated as illustrated in Equation 2-35.

Equation 2-35

$$B_{1,1} = (2+1+3+2)/4 = 2.00$$
$$B_{1,2} = (1+2+2+3)/4 = 2.00$$

The centroids of the remaining three clusters are calculated in the same manner and the results are tabulated in Table 2-8, and the initial partitioning of the sample space is illustrated in Figure 2-14.

Table 2-8: K-Means Example – First Iteration Cluster Means

Cluster	Contents	Cluster Means	
		x_1	x_2
1	A E F G	2.00	2.00
2	B C L	5.00	2.67
3	D H I	7.00	2.67
4	J K	6.50	4.50

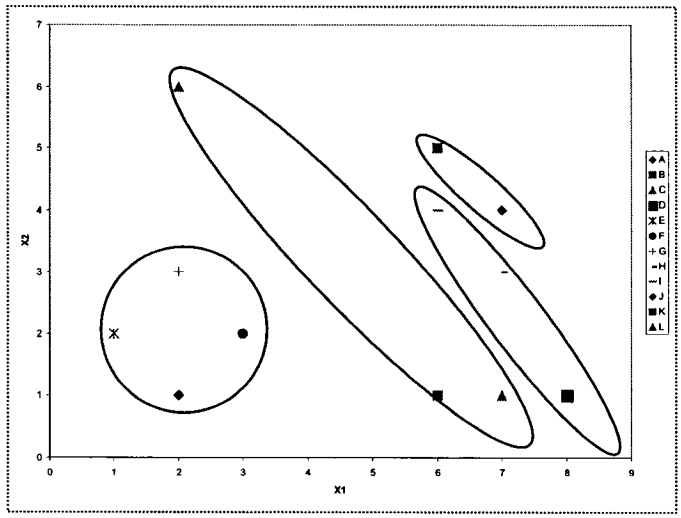

Figure 2-14: K-Means Example – Initial Sample Space Partitioning

The following step involves the calculation of the overall error for this classification iteration given by Equation 2-31. The results are shown in Equation 2-36.

Equation 2-36

$$\begin{aligned}\varepsilon &= (2-2)^2 + (1-2)^2 + (6-5)^2 + (1-2.67)^2 + (7-5)^2 + (1-2.67)^2 \\ &+ (8-7)^2 + (1-2.67)^2 + (1-2)^2 + (2-2)^2 + (3-2)^2 + (2-2)^2 + (2-2)^2 \\ &+ (3-2)^2 + (7-7)^2 + (3-2.67)^2 + (6-7)^2 + (4-2.67)^2 + (7-6.5)^2 + \\ &(4-4.5)^2 + (6-6.5)^2 + (5-4.5)^2 + (2-5)^2 + (6-2.67)^2 \\ &= 42.35\end{aligned}$$

Attempts must now be made to reduce the error. The algorithm proceeds by examining each object in turn, and calculating the effect of transferring that object to a different cluster. For instance, for the first object 'A', the squared Euclidean distance to each cluster center is calculated and the corresponding change in error, $\Delta\varepsilon$, is determined for moving object 'A' from its original cluster to each of the other clusters in the sample space as tabulated in Table 2-9.

Table 2-9: K-Means Example – Second Iteration Object A

Cluster 1	$D_{A,1}^{\ 2} = (2.00-2.00)^2 + (1.00-2.00)^2 = 1.00$
Cluster 2	$D_{A,2}^{\ 2} = (2.00-5.00)^2 + (1.00-2.67)^2 = 11.79$ $\Delta\varepsilon = (3)(11.79)/4 - (4)(1)/3 = 7.51$
Cluster 3	$D_{A,3}^{\ 2} = (2.00-7.00)^2 + (1.00-2.67)^2 = 17.79$ $\Delta\varepsilon = (3)(17.79)/4 - (4)(1)/3 = 19.51$
Cluster 4	$D_{A,4}^{\ 2} = (2.00-6.50)^2 + (1.00-4.50)^2 = 32.50$ $\Delta\varepsilon = (3)(32.50)/4 - (4)(1)/3 = 20.34$

Examining the $\Delta\varepsilon$ values for object 'A' in Table 2-9 indicates that they are all positive; relocating object 'A' to another cluster would only serve to increase the overall error. Visually examining Figure 2-14 indicates that object 'A' is closest to the center of Cluster 1, and nothing would be gained by relocating it. This process is repeated for every object, *i*. From Figure 2-14 it can be observed that object 'C' would be closer to the centroid of cluster 3 rather than cluster 2. Table 2-10 illustrates the effect of moving object 'C' from cluster 2 to each of the other clusters.

Table 2-10: K-Means Example – Second Iteration Object C

Cluster 2	$D_{C,2}^{\ 2} = (7.00-5.00)^2 + (1.00-2.67)^2 = 6.79$
Cluster 1	$D_{C,1}^{\ 2} = (7.00-2.00)^2 + (1.00-2.00)^2 = 26.00$ $\Delta\varepsilon = (4)(26.00)/5 - (3)(6.79)/2 = 3.82$
Cluster 3	$D_{C,3}^{\ 2} = (7.00-7.00)^2 + (1.00-2.67)^2 = 2.79$ $\Delta\varepsilon = (3)(2.79)/4 - (3)(6.79)/2 = -14.88$
Cluster 4	$D_{C,4}^{\ 2} = (7.00-6.50)^2 + (1.00-4.50)^2 = 12.50$ $\Delta\varepsilon = (2)(12.50)/3 - (3)(6.79)/2 = -8.64$

Relocating object 'C' from cluster 2 to cluster 3 decreases the overall error by 14.88; therefore, object 'C' can be relocated from cluster 2 to cluster 3, and the overall system error can be recalculated as shown in Equation 2-37.

Equation 2-37

$$\varepsilon = 42.35 - 14.88 = 27.47$$

The new clusters and cluster centers with object 'C' relocated are calculated and tabulated in Table 2-11.

Table 2-11: K-Means Example – Second Iteration Cluster Means

Cluster	Contents	Cluster Means	
		x_1	x_2
1	A E F G	2.00	2.00
2	B L	4.00	3.50
3	C D H I	7.00	2.50
4	J K	6.50	4.50

On the next pass, object 'B' is transferred from cluster 2 to cluster 3. The cluster populations and their newly calculated centers are tabulated in Table 2-12.

Table 2-12: K-Means Example – Third Iteration Cluster Means

Cluster	Contents	Cluster Means	
		x_1	x_2
1	A E F G	2.00	2.00
2	L	2.00	6.00
3	B C D H I	6.80	2.00
4	J K	6.50	4.50

On the next pass, object 'I' is relocated from cluster 3 to cluster 4 as tabulated in Table 2-13.

Table 2-13: K-Means Example – Fourth Iteration Cluster Means

Cluster	Contents	Cluster Means	
		x_1	x_2
1	A E F G	2.00	2.00
2	L	4.00	6.00
3	B C D H	7.00	1.50
4	I J K	6.33	4.33

On the next pass, object 'H' is relocated from cluster 3 to cluster 4 as tabulated in Table 2-14.

Table 2-14: K-Means Example – Fifth Iteration Cluster Means

Cluster	Contents	Cluster Means	
		X_1	x_2
1	A E F G	2.00	2.00
2	L	4.00	6.00
3	B C D	7.00	1.00
4	H I J K	6.50	4.00

The process is repeated a final time, and no movement of any object between clusters yields a better result. Figure 2-15 depicts the final cluster arrangement.

Although a value of K = 4 for the number of clusters was arbitrarily chosen, examination of the data indicates that values for K of 2 or 3 could have been used as well. Cluster analysis is not considered a statistical test, and therefore the choice or criteria for the best results are always at the discretion of the analyst (Adams, 1995).

K-Means clustering is considered a "hard" clustering method; all of the objects within a cluster are weighted equally. A borderline object that may rest equally between two clustering groups will only contribute to the mean of the group in which it was assigned. Similarly, an outlier may have a drastic effect on the outcome of the K-Means algorithm (MacKay, 2003).

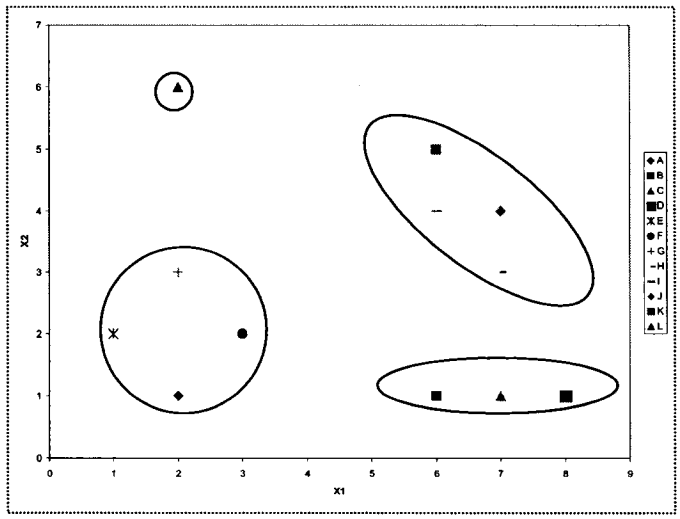

Figure 2-15: K-Means Example – Final Clusters

2.7 Genetic Programming

Genetic programming is a technique used to generate and optimize a desired computational function based on the concepts of Darwinian selection (See: *The Origin of Species on the Basis of Natural Selection*, Darwin). An initial random population of individuals, each encoding a computational function, is generated. The fitness of these individuals is evaluated and assessed on the basis of obtaining the desired output. New individuals, or offspring, are produced by *mutation* (the introduction of one or more random changes in the composition of the parent individual) or by *crossover* (randomly rearranging functional components between two or more parents). The fitness of the new individuals is then assessed. The individuals from the total population with the highest fitness level are selected to be the parents of the next generation. The process is repeated until the desired result is obtained, or the rate of improvement in the population becomes zero. Research has shown that the genetic method can approach the theoretical optimum efficiency of a search algorithm (Gilbert et al., 1997).

2.7.1 Fitness Function

The *fitness function* is a measure of the success of evolution for the genetic algorithm. The fitness function is completely dependent on the goal of the genetic program. A fitness function is a performance measure, or reward function, and has the greatest analogy to natural selection (Russell and Norvig, 1995).

Typically, the reward function is an algorithm that takes an individual as input, evaluates the result of a process versus the desired result, and outputs a real number score based on the individual's fitness. It is these scores that are assessed when determining the parents for the next generation.

It is common in genetic algorithms that incorporate other analytical procedures, such as neural networks, to have a penalty function associated with the fitness function. In other words, if an extremely large neural network has the same output as a smaller network, then the smaller one will be assigned a higher fitness function as the larger one will be penalized at a constant multiplied by the number of nodes (Gilbert et al., 1997).

Considerations in the evaluation of criteria with a fitness function should include the selection of the training and validation sets of data. By randomly assigning samples to these two sets on every evaluation, the problem of over-training to any one set is avoided (Gilbert et al., 1997).

2.7.2 Selection

As the evolutionary process learns via a fitness function, i.e. its rewards are its offspring, then the genetic algorithm can be seen as a form of reinforcement learning. However, no attempt is made to learn the relationship between the rewards or the actions taken by the agent. Genetic algorithms simply search the sample space with the goal of

finding an individual, or individuals, that maximize the fitness function (Russell et al., 1995).

Genetic algorithms being search algorithms, or hill climbing algorithms, must take care not to get stuck on local maxima or minima when attempting to produce the desired optimum individual. Therefore, individuals with low-scoring fitness functions cannot be ignored. Typically on a set interval (i.e. every 10 generations), a randomly selected lower scoring individual is reintroduced into the parent population for the next generation (Gilbert et al., 1997).

2.7.3 Architecture

Prior to applying the genetic algorithm to a problem, certain questions must be addressed in order to produce a rugged architecture (Russell et al., 1995):

- What is the fitness function?
- How is an individual represented?
- How are individuals selected?
- How do individuals reproduce?

In the biological genetic makeup, an individual gene is represented by a string of characters from a finite alphabet (A, G, T, C), where each element of the alphabetic string represents a nucleic acid (adenine, guanine, thymine, cytosine). In genetic algorithms on the other hand, individuals are usually represented by the binary alphabet (0, 1). These bits are represented in a bit string (Russell et al., 1995).

Typically, selection strategies are randomized based on the probability of selection as a function of fitness function. For example, if individual X scores twice as high as individual Y with the fitness function, then X is twice as likely to be chosen as the parent for the next generation. The randomized selection process is typically selected from only the top 30% of the population. Usually, selection is done with replacement, such that a strong individual will get to reproduce several times.

Reproduction is accomplished by crossover and mutation. The individuals selected for reproduction are randomly paired. For each pair, a random crossover point is chosen, and the first part of the first individual (up to the crossover point) is paired with the second part of the second individual to produce one offspring. Conversely, the second part of the first individual is paired with the first part of the second individual to produce the second offspring. However, each offspring gene is subject to a small independent probability of mutation, where a bit is randomly selected and converted from 1 to 0 or vice versa. Figure 2-16 illustrates the reproduction stage of the genetic algorithm.

The genetic algorithms is typically programmed to stop when the desired fitness level is reached or the rate of improvement falls to a low level or a specified number of maximum generations has been achieved.

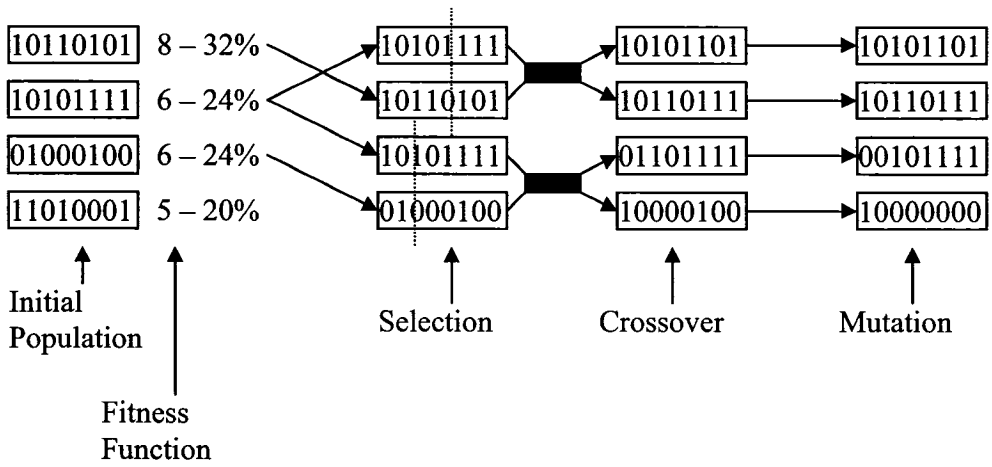

Figure 2-16: Reproduction Stage of a Genetic Algorithm

Genetic algorithms are relatively easy to apply to a wide range of analytical problems. On some problems the results can be excellent, and poor on others.

2.8 Outlier Removal

The presence of outliers or rogue values consistently causes problems for analysts. Analysts must be able to not only detect outliers but also develop some method of systematically reducing their effects on the end results (Adams, 1995).

A common method for mathematically detecting outliers is to compare the difference between the observed value and some expected, predicted or modeled value (referred to as a residual). By calculating the standard deviation of all of the residuals in the data set, and determining a threshold (i.e. a multiple of the standard deviation), then the sample may be rejected if it falls outside of those conditions (Adams, 1995). If a sample is rejected, then it can be completely removed and its value discarded.

3 Using Genetic Algorithm to Optimization of Pre-Processing Variables

3.1 Introduction

The recent declassification of military technology has made available the infrared focal plane array (FPA), which has introduced a new dimension of FT-IR spectroscopy (Wolsky, 1989). The FPA allows for geometrical information to be captured together with spectral information (Van Den Broek et al., 1997). In addition, this newer technology lends itself to improved acquisition speeds that also add to the allure of this imaging technique.

The focal plane array consists of thousands of sensing elements or "pixels", each capable of capturing a complete spectrum (Van Den Broek et al., 1997). With the added geometric dimensionality, some of these pixels may not be collecting spectral information about the sample under examination. As well, due to the manufacturing process and quality control some pixels, typically less than one percent, may not be functioning at all and therefore only producing noise. Manufacturing a flawless superconductor such as a FPA is extremely rare and difficult (Wolsky, 1989). Therefore it is imperative to carefully consider the selection of pixels to be utilized for further analysis.

Like traditional infrared spectroscopy, infrared imaging has found application in the classification of samples based on their infrared spectral characteristics. Again, selection of appropriate pre-processing variables is essential for the development of

accurate and reproducible classification models. Classification functions rely on pre-processing algorithms to enhance the spectral data and to select and extract features relevant to effective segmentation. Furthermore, pre-processing algorithms help to amplify the spectral regions essential to the differentiation of one classification group from another.

The plethora of data contained in the spectral images acquired from a FPA makes processing computationally expensive. Manual manipulation of pre-processing variables can be accomplished based on some familiarity with the data and the personal experience of the analyst (Jarvis and Goodacre, 2004). There are many different pre-processing algorithms to choose from, making the process of informed trial and error inherently complex. In some cases, researchers will co-add the signals from all of the pixels in an image in order to produce an average spectrum and hence reduce the computational expense as they attempt to find an ideal processing sequence.

One could identify a selection of pre-processing variables for analysis and then evaluate every possible combination applied to the training data; however, this would be computationally exhausting and time intensive. The computational expense is an exponential function of the size of the data set (Jarvis et al., 2004). In order to explore the search space efficiently, a heuristic search algorithm can be applied to find sub-optimal solutions (Russell, 1995).

The aim of this study is to examine the utilization of a heuristic search algorithm, namely the genetic algorithm, to optimize the segmentation of untreated, raw, infrared image data acquired from randomly selected foodborne bacterial cultures into Gram-positive and Gram-negative categories. While the effectiveness of the clustering is crucial

to the success of the algorithm, the secondary aim of the study is to utilize the same algorithm to conserve as much of the original pixel data as possible, while not discarding them during the data enhancement and selection procedures.

3.2 Materials and Methods

3.2.1 Organism growth, preparation and spectral acquisition

One hundred and eighty-seven previously identified and confirmed foodborne bacterial cultures were selected and acquired from both Health Canada and the US Food and Drug Administration (Kirkwood et al., 2004). The cultures were maintained at -86°C. Prior to spectral acquisition, bacteria were streaked onto Universal Media™ agar plates (Quelab Inc., Montreal, Canada) and then cultured for 16-18 hours. Bacterial colonies scraped from the agar plates were then deposited in triplicate, with no pre-treatment or staining, onto an infrared-transparent ZnSe slide. Each sample occupied an area of approximately 1 mm^2 , allowing more than 200 samples to be deposited on the same slide (Kirkwood, 2004). Each slide was allowed to air dry for about 10 minutes in order to a produce a film for infrared analysis.

Infrared images were acquired in triplicate using a Varian Excalibur (Varian, Randolph, MA) imaging spectrometer equipped with a UMA-600 infrared microscope and a liquid-nitrogen-cooled mercury cadmium telluride (MCT) focal-plane-array detector comprising 32 × 32 (1024) pixels. Using a 15× Schwarzschild objective, the field of view of the microscope was 176 × 176 μm. The imaging spectrometer was constantly purged with dry air to reduce the spectral contributions of atmospheric carbon dioxide and water vapor. Each image consisted of 256 co-added scans at a resolution of 8

cm^{-1}. Once collected, each image was divided by a background image to produce absorbance values.

Once the samples were cataloged and indexed, a random number generator was used to randomly select twelve Gram-positive and twelve Gram-negative samples from the collection to be used as a training set for the classification algorithm. The samples selected are listed in Table 3-1.

Table 3-1: Training Set - Bacterial Species and Specimens

Strain Name	Specimen #	Culture #	Gram-
Klebsiella oxytoca	112	2	Negative
Hafnia alvei	115	2	Negative
Klebsiella pneumoniae	145	2	Negative
Shigella flexneri	165	4	Negative
Escherichia coli	1125-26	2	Negative
Escherichia coli	1156-2	3	Negative
Salmonella berta	804	2	Negative
Salmonella derby	4359	1	Negative
Salmonella derby	4359	2	Negative
Escherichia coli 0157:H7	149	1	Negative
Salmonella heidelberg	3221	1	Negative
Salmonella heidelberg	3221	2	Negative
Streptococcus xylosus	244	2	Positive
Streptococcus xylosus	244	3	Positive
Clostridium sporogenes	241	1	Positive
Listeria monocytogenes	4404	1	Positive
Listeria monocytogenes	4410	1	Positive
Listeria monocytogenes	4410	2	Positive
Listeria monocytogenes	4426	1	Positive
Listeria monocytogenes	4749	2	Positive
Staphylococcus aureus	251	4	Positive
Listeria monocytogenes	688	3	Positive
Listeria monocytogenes	1116-2	1	Positive
Listeria monocytogenes	1116-2	2	Positive

3.2.2 Constructing the Genetic Algorithm

The genetic algorithm is the heuristic search algorithm used to sub-optimize the combinations of pre-processing variables. In order to examine the effectiveness of a genetic algorithm for the optimization of the segmentation of infrared image data, careful

consideration must be made to the variable selection. Because the addition of each variable to the algorithm increases the computational cost, a group of ten commonly used pre-processing techniques were selected to be evaluated in this research.

The ten pre-processing techniques are represented digitally by a 27-bit binary string or DNA series of pre-processing variables as illustrated in Figure 3-1. The 27-bit binary string signifies 134,217,728 possible combinations of pre-processing variables. Although there are many possible combinations, further examination of the flow chart reveals that some pre-processing variables are interdependent. Consequently, there are only 28,753,920 unique combinations of pre-processing variables.

The study was authored in Matlab R13 (Mathworks, Boston, MA) and executed on a personal computer with an AMD Athalon 64 3400+ CPU, 1 gigabyte of RAM. The operating system was Windows XP Professional x64.

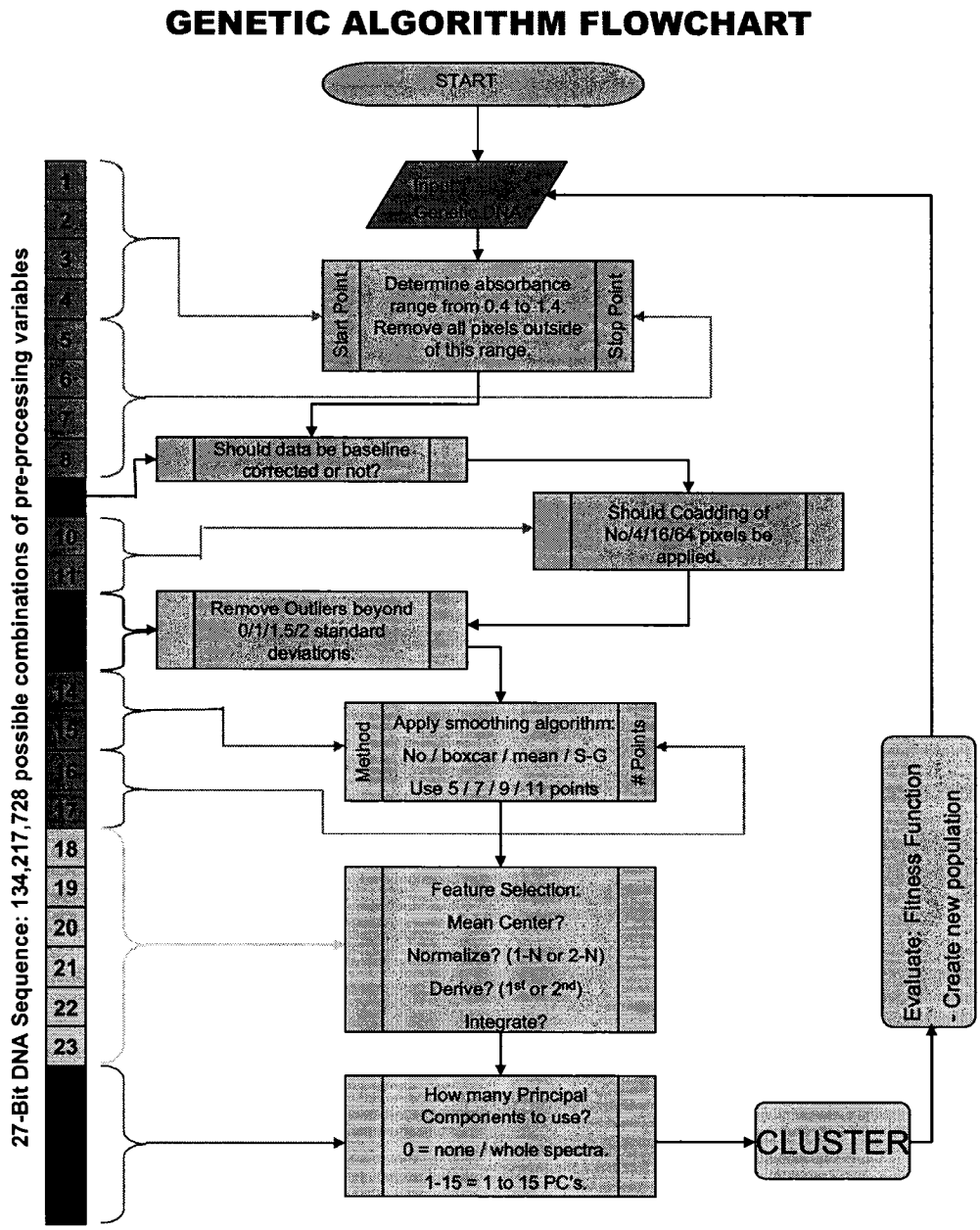

Figure 3-1: Genetic Algorithm Flow Chart

The first pre-processing variable evaluated during each iteration of the genetic algorithm is the acceptable absorbance range of the protein amide I band, which is used as a measure of sample thickness since all bacterial cells contain protein. Within each of

the 24 images selected from Table 3-1 each of the 1024 pixels is evaluated at a wavenumber of 1650 cm^{-1} with a two-point baseline correction at 1780 cm^{-1} and 980 cm^{-1}. The first eight bits of the 27-bit binary DNA string represent an acceptable numerical absorbance range from 0.4 to 1.4 in increments of 0.066. If the absorbance at 1650 cm^{-1} is outside of the defined range, then the pixel data is discarded.

The second pre-processing variable evaluated is whether to use the baseline corrected data or the non-baseline corrected data for the future pre-processing procedures and the eventual clustering. The ninth bit in the DNA string represents a binary on/off switch. If the value is zero, the non-baseline corrected image data is used. If the value is one, the baseline corrected image data is used.

The third pre-processing variable evaluated is whether or not to co-add some of the adjacent pixels within an image to reduce the noise. The twelfth and thirteenth bits in the DNA string represent four possible co-adding solutions. The first is not to co-add the data at all, and go on to the next pre-processing step. The second through the fourth are to co-add 2 × 2 (4), 4 × 4 (16) or 8 × 8 (64) adjacent pixels. The new co-added pixels become the data that is passed on to subsequent procedures. Because only adjacent pixels are co-added and some of the image data was already discarded in the first pre-processing step, left-over or orphaned pixels are discarded.

The fourth pre-processing variable evaluated is whether or not to remove outlier data points. The tenth and eleventh bits in the DNA string represent four possible outlier removal solutions. The first solution is not to remove any outliers, and the data is passed to the following pre-processing procedure. The second to fourth solutions require that within each of the 24 images, the remaining pixel or co-added data is averaged and a

standard deviation is calculated from the mean. Pixels falling outside 1, 1.5, or 2 standard deviations are discarded.

The pre-processing variables dealt with up to this point are concerned primarily with pixel selection. The following pre-processing variables are concerned primarily with data manipulation and feature selection.

The fifth pre-processing variable is represented in the DNA string by bits fourteen through seventeen. The first two bits dictate if a smoothing algorithm should be applied, and if so which one. The procedure offers boxcar, mean and Sarvitzky-Golay smoothing algorithms. The last two bits represent the number of data points to use in conjunction with the smoothing algorithm (5, 7, 9 or 11). If no smoothing is selected, than these two bits are ignored.

The sixth pre-processing variable evaluated is whether or not to mean center the data. The eighteenth bit in the DNA string is a binary on/off flag. If the flag is set to on, the mean of all data is subtracted from each sample.

The seventh pre-processing variable evaluated is whether or not to normalize the data. The nineteenth bit in the DNA string is the binary on/off flag. If the flag is set to on, the twentieth bit dictates which normalization method to apply, normalization to unit area or normalization to unit length.

The eighth pre-processing variable evaluated is whether or not to take the derivative of the data. The twenty-first bit in the DNA string is the binary on/off flag. If the flag is set to on, the twenty-second bit dictates whether to take the first or second derivative of each of the individual spectra in the data set.

The ninth pre-processing variable evaluated is whether or not to integrate the data. The twenty-second bit in the DNA string is the binary on/off flag. If the flag is set to on, each of the individual spectra is integrated.

The tenth and final pre-processing variable evaluated prior to clustering is the selection of principal components. The four bits from twenty-four through twenty-seven numerically represent a number from zero to fifteen. A value of zero passes the data without selecting principal components. Values from one to fifteen represent the number of principal components to pass on to the clustering algorithm.

Once the data has been passed through the pre-processing variable combination outlined in a particular DNA string, a portion of the data within each image has been discarded and the remaining pixels have been subjected to manipulation and feature selection. The remaining pixels are then segmented using the K-means natural clustering algorithm and the squared Euclidean distance metric. The algorithm is programmed to restart 5 times and take the best result in order to avoid local minima. The resulting clusters are compared with the known Gram-positive and Gram-negative classification of the samples in the training set.

The fitness function scores the accuracy of the iteration by incorporating the clustering accuracy and the number of remaining pixels after all of the pre-processing has been accomplished and assigns it a score. For the purpose of this study, the classification accuracy was assigned a weight of 80% and the pixel conservation (number of spectra at clustering / number of original spectra) a weight of 20% resulting in a score from 0 to 1.

In the first few iterations of the algorithm, DNA strings are generated randomly. After fifty iterations have been reached, the results of each iteration are sorted in order of

their fitness function score. New DNA strings are created by assigning exponential weight to the higher scoring DNA strings and selecting two existing DNA strings to become the parents of the subsequent generations. The two selected parents are then split at a random bit and cross-bred to produce two unique children. At random occurrences, a genetic mutation will be randomly applied to one of the children. In some iterations a single parent may be bred with a randomly generated parent. The frequencies of the random mutations and random parent introductions are less than 5% of the time. Each generation produces two offspring DNA strings. The DNA strings are then subjected to the algorithm evaluated by the fitness function and assigned a place within the population.

3.3 Results and Discussion

3.3.1 Overview: Arrival at Sub-Optimal Solution

The data from the infrared images in Table 3-1 were input into the genetic algorithm and it was allowed to run. Each iteration of the algorithm increased in fitness until a sub-optimal solution was discovered after 316 iterations as illustrated in Figure 3-2. Each iteration took on average 76 seconds to execute. The sub-optimal solution was discovered in approximately six hours and 40 minutes of unsupervised processing.

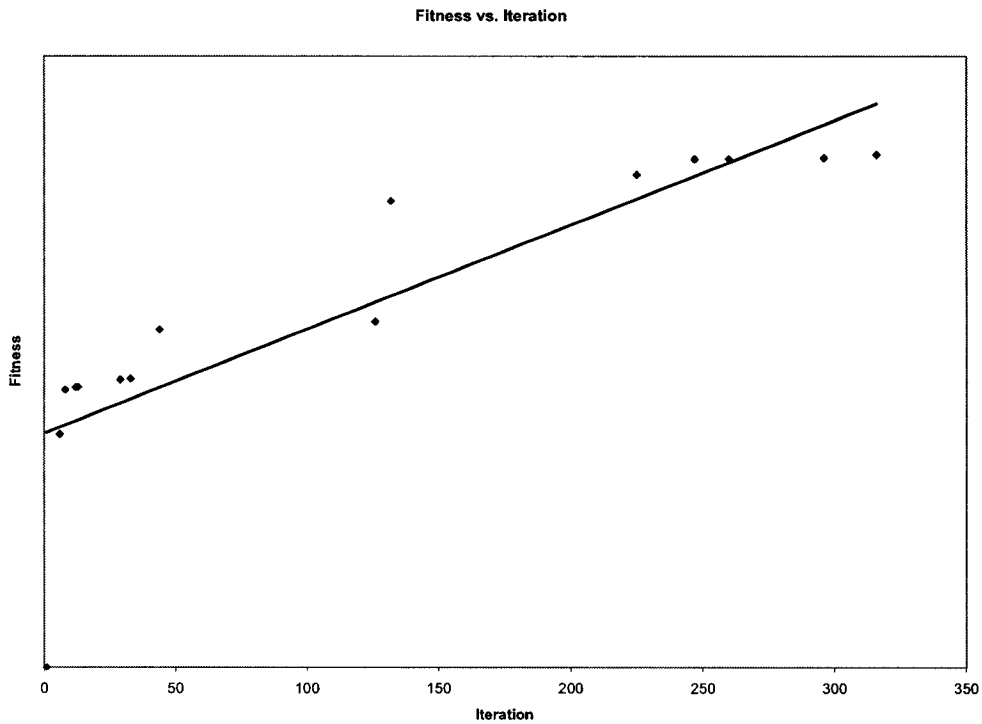

Figure 3-2: Genetic Algorithm - Fitness vs. Iterations

The linear regression line superimposed on the data in Figure 3-2 illustrates the increase in fitness, or optimization of pre-processing variables with respect to the number of iterations for this particular data set. The sub-optimal solution represents a fitness score of 84.0%. The majority (80%) of the fitness score is comprised of the 97.8% accuracy achieved for the clustering of the data for the Gram-positive and Gram-negative samples and the balance (20%) is comprised of the utilization of 28.6% of the available data (7,029 data points). Figure 3-3 illustrates the processes determined by the sub-optimal solution.

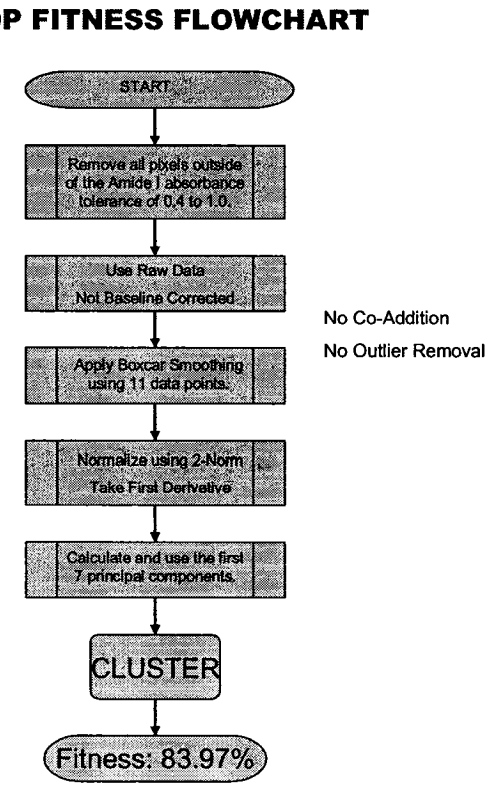

Figure 3-3: Flowchart of Sub-Optimal Fitness DNA

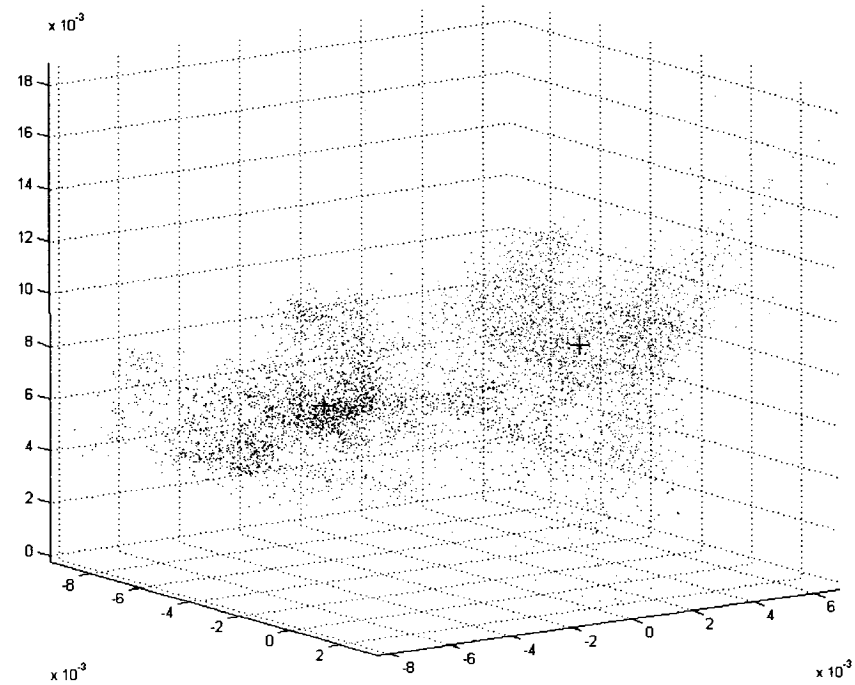

Figure 3-4: Genetic Algorithm - 3-D Projection of Top Fitness Member

Figure 3-4 is an illustration of the clusters projected in three-dimensional space. The crosses represent the centroid of each cluster. The red dots represent correctly classified pixels in the images of the Gram-negative samples while the green dots represent correctly classified pixels in the images of the Gram-positive samples. The blue dots represent incorrectly classified pixels.

Following iteration 316 where the sub-optimal solution was identified, the algorithm was allowed to continue until it reached 2,100 iterations. The 2,100 iterations represent 0.0073% of the possible unique combinations of pre-processing variables.

Although the maximum fitness was discovered at iteration 316, it did not correspond to the maximum clustering accuracy. Iteration number 1,885 revealed a nearly perfect classification accuracy of 99.8% as opposed to the sub-optimal value of

97.8%. The difference in the fitness score is attributed to the 20% weight of the number of data points used. Iteration 1,885 utilized 3,293 data points to build the classification model while the sub-optimal fitness utilized 7,029.

Efforts were made within the genetic programming to avoid local maxima within the state space by introducing randomized strings and mutations (Russell, 1995). Undoubtedly when exploring such a massive state space with such a limited numbers of iterations, DNA generations are likely to converge on a sub-optimal solution with similar attributes.

3.3.2 Acceptable Amide I Absorbance Range

The top fitness performer utilized an acceptable amide I tolerance range of 0.4 to 1.0 absorbance units with the two four-bit DNA segments in Table 3-2. Each of the eight-bit strings represents two four-bit absorbance boundaries. The pre-processing module ensures that the boundaries are utilized in numerical order. The acceptable absorbance range pre-processing module interprets the four-bit binary strings by converting them to their decimal value, multiplying the number by 6/90 and adding 0.4.

Table 3-2: Amide I Absorbance Tolerance

Boundary:	1	2
Segment:	0000	1001
Value:	0.4	1.0

Examination of the top fitness performers in the population reveals relevant information pertaining to the data set. The acceptable amide I ranges for the top fifty fitness performers of the population are plotted in Figure 3-5. The top seven performers all share the same amide I range of 0.4 to 1.0. The majority of the top performers have a tolerance range that begins at 0.4 and ends at 0.8.

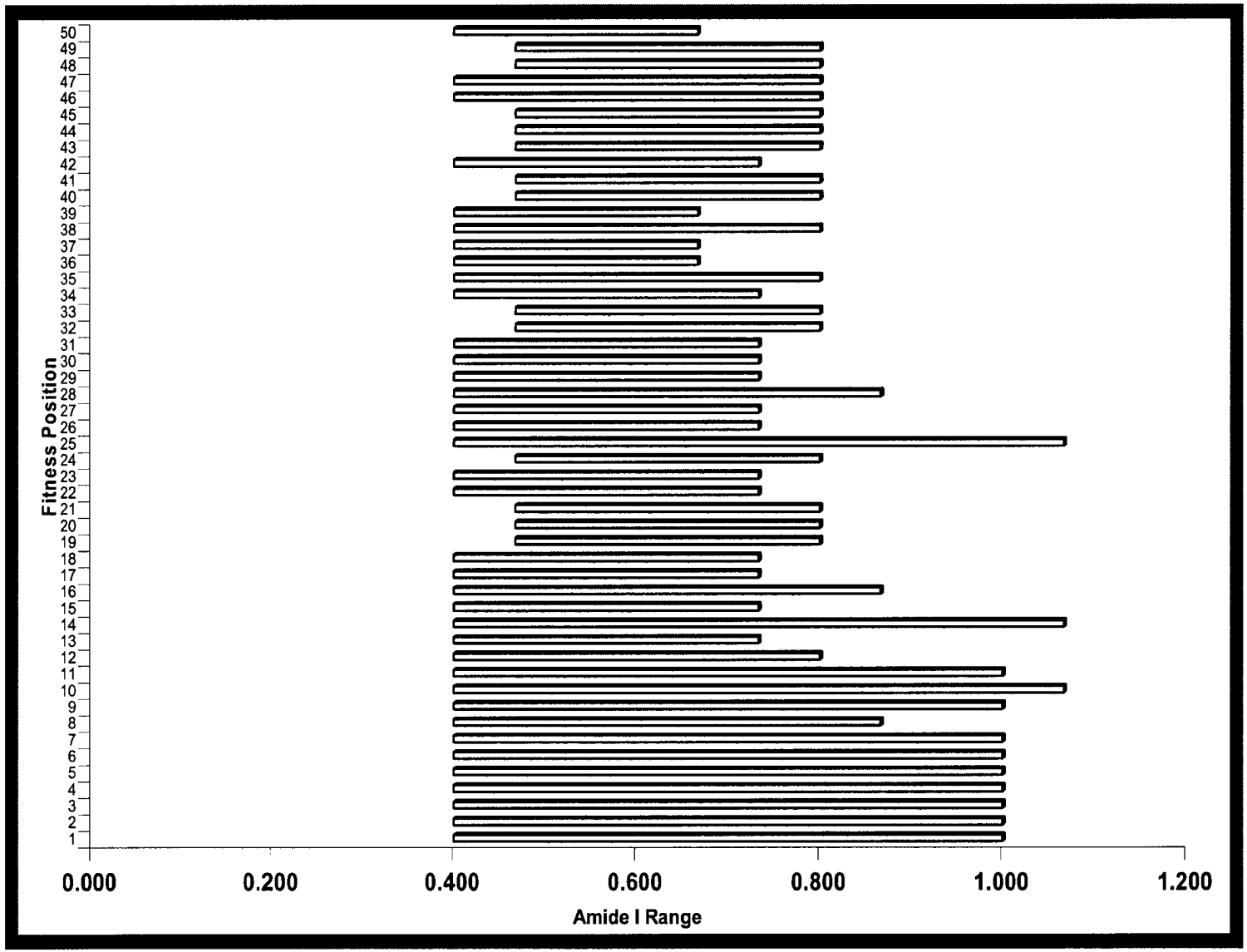

Figure 3-5: Amide I Absorbance Tolerances for Top 50 Fitness Population

At 1,024 pixels per image and 24 images, the initial data set comprised 24,576 individual spectra. Once subjected to the boundary values of this pre-processing module, the number of retained pixels in the sub-optimal solution dropped to 7,029 – discarding 71.4% of the original pixels. Figure 3-6 illustrates the pixel usage of the Escherichia coli image before and after pixels outside the boundary values were discarded.

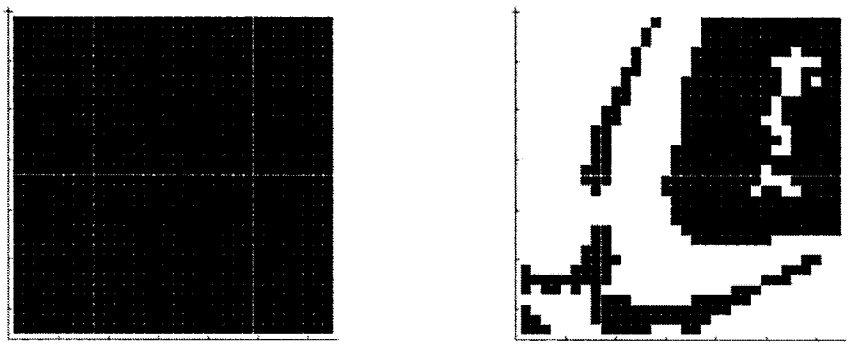

Figure 3-6: Pixel Utilization Before and After Selection Based on Amide I Tolerance

There can be several explanations for the selection of these particular boundary values by the genetic algorithm. One such is that the sample material is not uniform on the slide. Areas where there is no smear at all would exhibit a uniform absorbance of approximately zero when the ratio was taken against a background. These pixels, not containing any useful data, would be excluded from the clustering algorithm. Similarly, any areas on the slide with extraneous material would not give rise to absorption of infrared energy at the wavenumber used to measure the amide I band. In contrast, in areas where the smear was excessively thick, causing most of the infrared energy to be absorbed, the spectral data would not be suitable for analysis.

Another potential contributor to the large amounts of data rejection may be the non-uniformity of the quantum sensitivity of the focal plane array from pixel to pixel

(Rainieri and Pagliarini, 2004; Davis, 2001; Adams, 1995). The particular boundary values selected by the genetic algorithm may correspond to the range of linear response. Dead pixels producing only random noise would probably fall outside of this tight tolerance range as well. Figure 3-7 is a contour map representing the sum of the instances of each pixel after being subjected to the amide I absorbance tolerances. The darkest blue color patterns represent pixels that were rejected in all of the 24 images. These pixels are probably the most likely candidates to be non-uniform or malfunctioning.

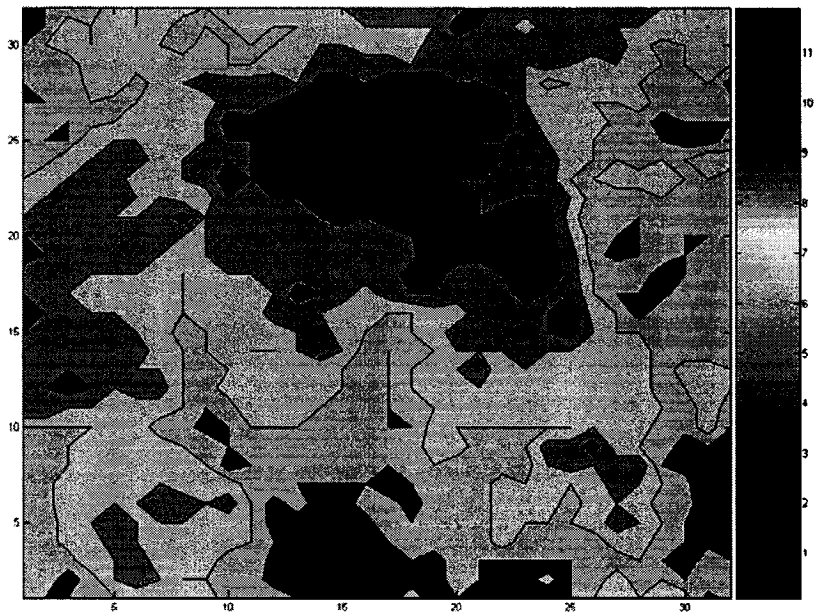

Figure 3-7: Average Pixel Utilization After Selection Based on Amide I Tolerances

3.3.3 Use of Baseline Corrected Data

The top fitness combination of pre-processing variables as well as the 49 runner-ups retained the raw data for processing as opposed to the baseline corrected data used to determine the amide I absorbance.

In order to verify that this was not solely due to a local maximum in the state space, the genetic DNA of the first sub-optimum solution was mutated by switching the ninth bit from 0 to 1 as illustrated in Tablo 3.3-2. This caused all further processing and clustering to be applied to the baseline corrected data as opposed to the raw data. When the baseline corrected data was used, the clustering accuracy dropped from 97.8% to 60.4%. This reduction in performance could be due to the combined use of the first derivative and the baseline correction causing essential identification features to be lost (Beebe et al., 1998).

Table 3-3: Baseline Correction Evaluation

Original DNA	0111010110011100000	0	00001001
Modified DNA 1	0111010110011100000	1	00001001

To test this possibility, the DNA string was altered once again to remove the derivative processing function as illustrated in Table 3-4. This instance produced an even further drop in clustering accuracy to 57.9%. Therefore, a local maximum in the state space did not dictate the arbitrary selection of the first derivative over the baseline correction pre-processing variables.

Table 3-4: Baseline Correction Evaluation – Derivative Mutation

Modified DNA 1	01110	1	011001110000	0	00001001
Modified DNA 2	01110	0	011001110000	1	00001001

Although the baseline corrected data is essential for evaluating the response of the amide I band, it is detrimental to the accuracy of the model for the sub-optimum classification solution.

3.3.4 Co-Addition of Pixels

None of the top 50 fitness DNA strings exhibited any pixel to pixel co-addition. Given the tight tolerances of the amide I absorbance response, the image pixels were

sparsely dispersed throughout the image as illustrated in Figure 3-6, which shows the pixel usage in the image of Escherichia coli. In some of the sample images, even attempting to co-add 2 × 2 adjacent pixels would be impossible, thereby eliminating the image from the clustering set. For future research, it may be beneficial to attempt to construct a genetic algorithm that would execute a co-addition module prior to evaluating acceptable absorbance ranges.

3.3.5 Outlier Removal

The genetic algorithm sub-optimized with not removing any outlier pixels. This stood true for the top seven fitness members of the population that also shared the same amide I absorbance tolerance of 0.4 to 1.0 absorbance units.

When examining the balance of the fifty top fitness combinations, it is apparent that there is a direct correlation between the acceptable amide I range and the tolerance for outlier removal. Table 3-5 illustrates the outlier tolerance selected by the genetic algorithm with reference to the acceptable amide I range.

Table 3-5: Outlier Removal Tolerances vs. Amide I Tolerances

Amide I Range		Outlier Tolerance
0.400	1.000	No outlier removal
0.400	0.867	>1.5 Standard deviations
0.400	0.667	No outlier removal
0.400	0.800	>1.5 Standard deviations
0.400	0.733	>1 Standard deviations
0.467	0.800	>1 Standard deviations

As the amide I tolerances were reduced and therefore limited the number of useable pixels, the outlier tolerance increased as well, resulting in the elimination of even more data. As the amide I tolerance is narrowed, the remaining pixels are more similar to one another than when the tolerance is greater. Because of the increased similarity and

the decreased number of data points, a higher outlier removal rate would even further increase the similarity between the remaining pixels and as a result produce a significantly higher clustering accuracy.

To examine the influence of a different outlier removal, the sub-optimal DNA string was modified to remove all outliers beyond only one standard deviation. The hypothesis is that this modification would result in the use of fewer pixels but increase the overall clustering accuracy. When this genetic code was executed, an additional 765 pixels were discarded, and the accuracy dropped to 67.5%. It is difficult to explain this phenomenon. It is possible that those pixels contained sample data essential to creating accurate clusters. Because the sample points sat beyond 1 standard deviation away from the mean, it is a justified argument to assume that they also were positioned at the furthest points of the clusters. Being located so far apart would further force the cluster centers to be separated by a greater distance, resulting in increased clustering accuracy. Figure 3-8 illustrates the Escherichia coli image before and after pixels were removed with the outlier tolerance of one standard deviation.

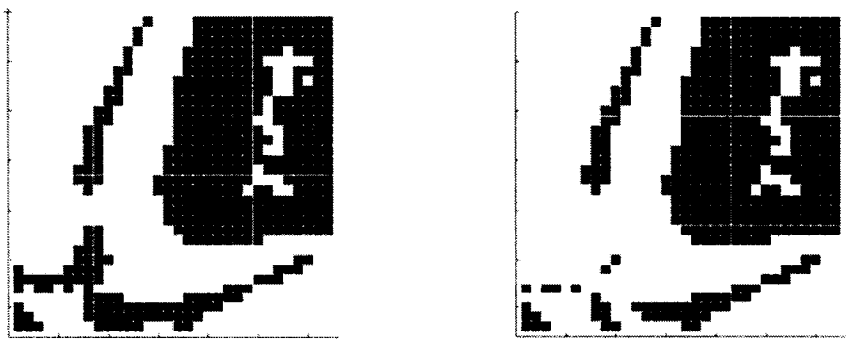

Figure 3-8: Pixel Utilization Before and After Outlier Removal

The pre-processing module charged with outlier removal has four possible functions including no outlier removal and removal of outliers beyond one, one and a half and two standard deviations from the mean. Examination of these results confirms that the outlier removal variable plays a significant role in the selection and conservation of spectral data for the eventual clustering accuracy of the data set. Future research would dictate that more increments for the outlier threshold to be optimized during the genetic algorithm search.

3.3.6 Smoothing

The DNA segment for the smoothing module exhibited a consistent and definite solution for all of the top fifty fitness performers. In every instance, the boxcar smoothing algorithm was applied with eleven smoothing points. As illustrated in Figure 3-9, the smoothing algorithm reduced the number of data points in each spectrum from 792 to 71 and smoothed the fluctuations within the signal. While the original image represented a data range of approximately 4000 cm^{-1} to 950 cm^{-1}, the smoothed image only represents a data range of approximately 3939 cm^{-1} to 968 cm^{-1} due to the end effects of the algorithm.

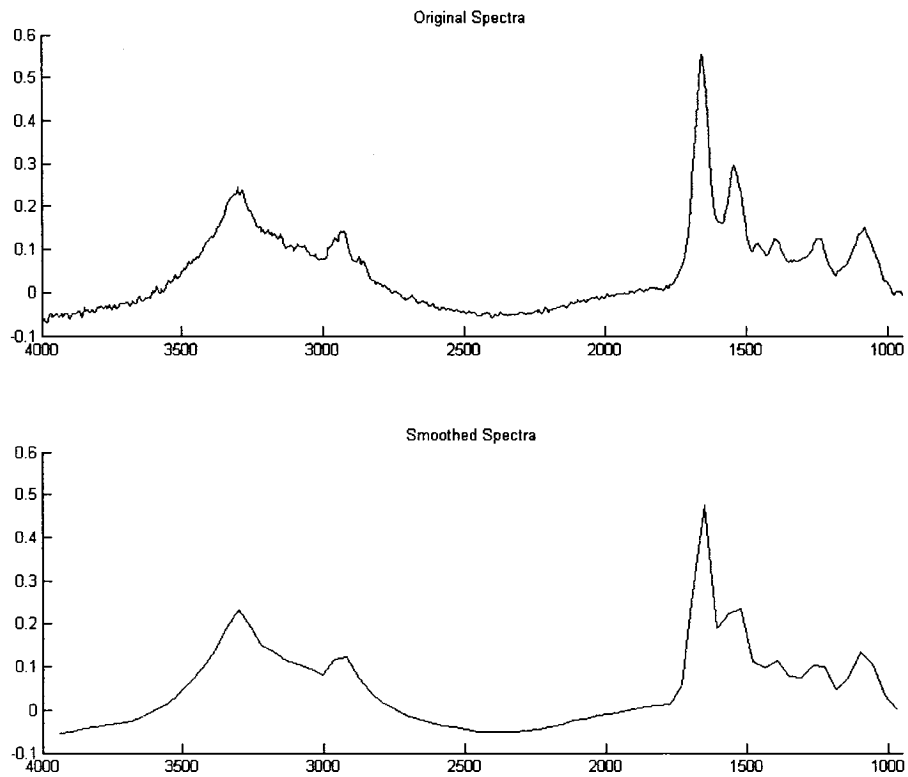

Figure 3-9: Smoothing Results – Absorbance vs. Wavenumber

In order to examine whether the smoothing method applied was due to a local maximum in the state space, the DNA strain was modified to execute the mean smoothing algorithm with eleven data points. This yielded a clustering accuracy of 65.4%. The strain was modified again to use the Savitzky-Golay method with eleven data points as well. This method yielded a clustering accuracy of 64.0%, which was significantly less than the sub-optimal solution. As the effects of Savitzky-Golay smoothing are similar to those of performing the first derivative, the DNA strain was modified again to remove the first derivative function. In this scenario, the clustering accuracy dropped to 51.4%.

To examine the effects of the number of data points used for smoothing, the procedure was repeated using the boxcar method with five, seven and nine data points. In these scenarios the clustering accuracy dropped to 61.2%, 67.9% and 63.9%, respectively. To examine the influence of smoothing, the algorithm was executed without applying any smoothing and resulted in a clustering accuracy of 64.9%.

In the state space where the sub-optimum solution is found, smoothing is an important step for noise reduction (Adams, 1995). The selection of the boxcar method ensures that the signal distortion caused by the smoothing algorithm is kept to a minimum while the number of smoothing points ensures that an adequate amount of signal noise is removed (Beebe et al., 1998). The Savitzky-Golay method inappropriately assigns weights to the smoothing points and over-distorted the signal. Yet, the derivative of the spectra remains important for proper classification.

3.3.7 Feature Selection

The feature selection processes are the functions that highlight the attributes to facilitate accurate clustering (Beebe et al., 1998). Of the available feature selection algorithms (mean centering, 1-normalization to unit area, 2-normalization to unit length, first derivative, second derivative and integration) the sub-optimum solution applied the 2-normalization and the first derivative. The normalization algorithm usually helps to compensate for differences in the sample volume (Beebe et al., 1998). The 2-normalization method normalizes each individual spectrum to unit length. Therefore, any variations in sample thickness during the sample preparation are compensated for by this algorithm (Adams, 1995). Of the top fifty fitness species, only fourteen used 1-normalization as opposed to 2-normalization. The rest of the feature selection variables

remained consistent. Figure 3-10 illustrates the effect of these two feature selection criteria on a sample spectrum.

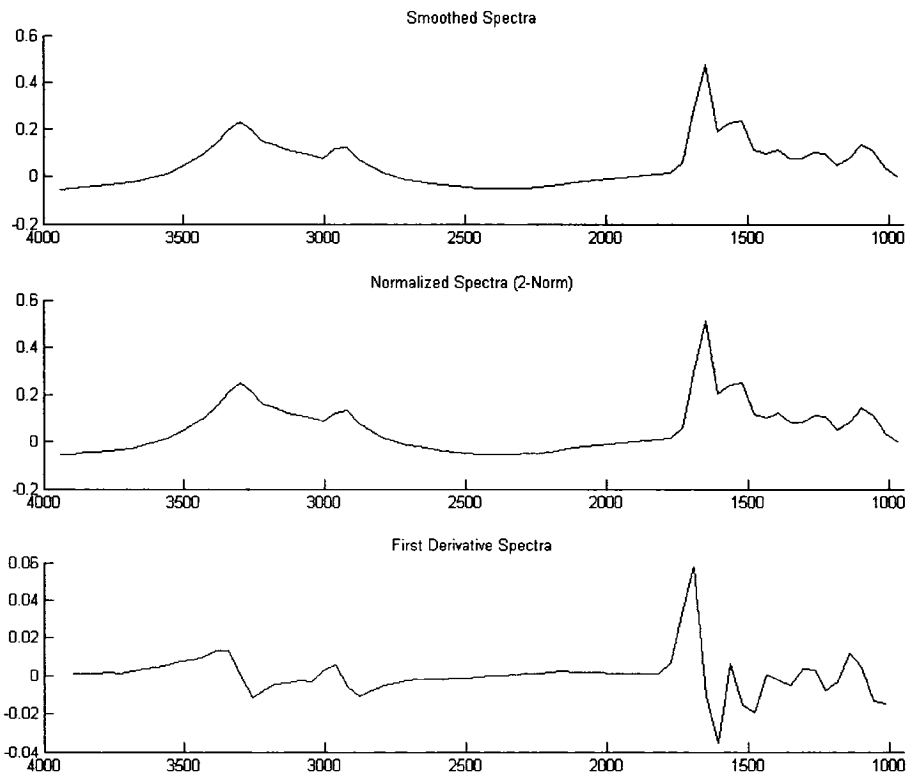

Figure 3-10: Feature Selection – Absorbance vs. Wavenumber

The first derivative typically reduces the effects of baseline fluctuation (Beebe et al., 1998). As discussed in Sections 3.3.3 and 3.3.6, the first derivative performed better than both the baseline corrected data and the Savitzky-Golay smoothing algorithm. The first derivative is independent of baseline corrections. There is no need to select a reference point or line in order to calculate the correction on a spectrum by spectrum basis. Therefore, the effects of spectral drift, or incorrectly choosing the baseline references, are not pertinent when performing the first derivative.

In order to ensure that none of the other feature selection protocols were overlooked due to local maxima in the state space, the DNA sequence was modified in the permutations listed in Table 3-6 to evaluate the influence of the other feature selection options on the top fitness species. While it would be lengthy to explore all combinations of feature selection protocols, it is apparent that the optimal feature selection criterion is in fact only the combination of normalization to unit length and taking the first derivative.

Table 3-6: Feature Selection Evaluation

									Accuracy
Original DNA	0111	0	1	0	1	1	0	0111000000000001001	97.8%
Mean Centering	0011	0	1	0	1	1	1	0111000000000001001	52.7%
No Normalization	0111	0	1	0	0	1	0	0111000000000001001	62.4%
1-Normalization	0011	0	1	0	1	0	0	0111000000000001001	57.8%
Second Derivative	0011	0	1	1	1	1	0	0111000000000001001	55.6%
Integrate	0011	1	1	0	1	1	0	0111000000000001001	60.8%

After accounting for the success of the normalization and first derivative, it is difficult to explain the inadequacies of the other feature selection processes within the state space. The negative effect of the mean centering could possibly stem from a combination of other pre-processing variables already compensating for the intercept of the data, and this algorithm over-compensating (Beebe et al., 1998).

Future considerations for this data set would be to remove the mean centering and integration variables to accelerate the search for a sub-optimal solution and allow for the computational expense to be spent on other pre-processing variables.

3.3.8 Principal Component Analysis

The sub-optimum fitness species utilized seven principal components to describe the data and accounted for 96.73% of the variance in the data. Table 3-7 illustrates the

variance accounted for by each of the principal components. Figure 3-11 illustrates the number of times within the top 50 fitness DNA strains that a number of principal components were utilized. It is apparent from the graph that the majority of the top performers utilized five to eleven principal components.

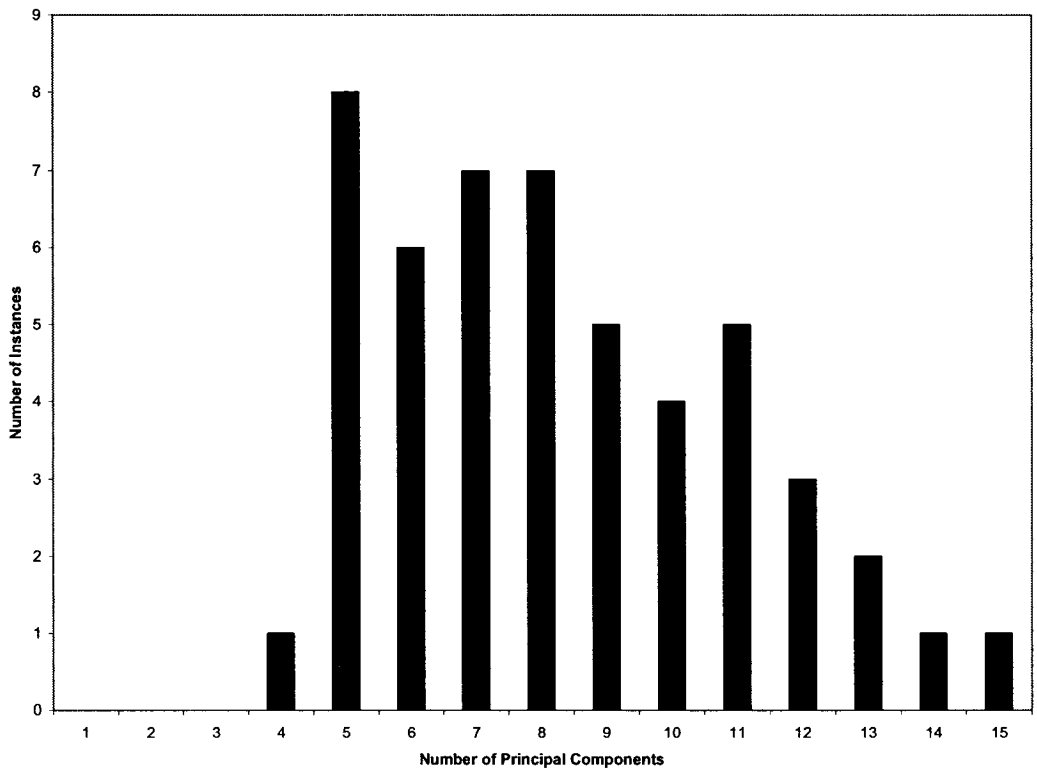

Figure 3-11: Number of Instances of Principal Components in Top 50 Fitness

Table 3-7: Principal Component Variance Contributions

Principal Component	Variance
1	76.86%
2	12.21%
3	2.23%
4	2.14%
5	1.69%
6	0.96%
7	0.64%

To evaluate the sub-optimum solution, the DNA was altered to allow for only three principal components accounting for 91.30% of the variance in the data. This solution resulted in a clustering accuracy of 91.6%. Moving in the other direction, the DNA was altered to use eleven principal components accounting for 98.4% of the variance. This solution yielded a clustering accuracy of 97.5%.

Using all of the data and no principal components yielded a clustering accuracy of 64.0%. In conjunction with increasing the number of principal components, it is apparent that a significant portion of the remaining data has a negative effect on the clustering accuracy, whereas using only a few principal components does not account for enough of the variance to accurately cluster the data

The plots of the linear weights of each of the seven individual principal components as well as their scaled by variance sum are illustrated in Figure 3-12. The knowledge that these seven principal components produce the most accurate clustering of the sample data helps to determine the spectral regions where the pertinent data is contained. Examination of the plot of the scaled sums of the first seven principal components yields two distinct spectral regions where the majority of the principal component weighting is applied. The first region lies between approximately 3684 cm^{-1} and 2750 cm^{-1} while the second region lies between approximately 1817 cm^{-1} and 1053 cm^{-1}.

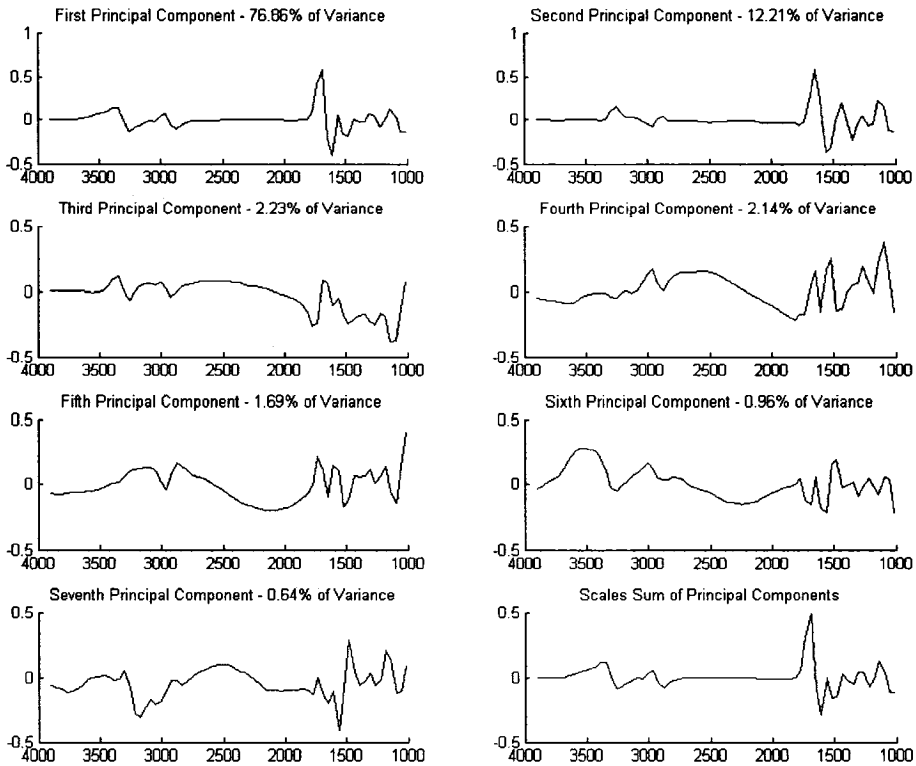

Figure 3-12: Principal Component Weights vs. Wavenumber

It is evident that the pertinent data is contained in the second region from 1817 cm^{-1} to 1053 cm^{-1}. In independent research on the entire data set of 200 samples, Kirkwood found that the region of 1770 cm^{-1} to 970 cm^{-1} contains the majority of the information required to cluster Gram-positive versus Gram-negative samples (Kirkwood, 2004). Due to the end effects of the smoothing algorithm, the leading data points were removed and hence we can conclude that the regions are almost identical (Adams, 1995).

3.4 Validation Tests

The successful sub-optimization of the pre-processing variables by the genetic algorithm separated the Gram-positive and Gram-negative bacterial samples in the

training set into segregated clusters. To test the reliability of this clustering model, and hence the appropriateness of the pre-processing variables selected, two validation tests were performed with samples not included in the training set. In the first validation test, replicate samples of the specimens included in the training set were classified. Subsequently, a second validation set consisting of other bacterial specimens was tested. In both cases, the infrared images of each of the selected samples were subjected to the pre-processing procedures determined by the sub-optimal solution obtained with the training set. The image data were thus treated as follows. Each of the images was baseline corrected using a two-point baseline at 1780 cm^{-1} and 980 cm^{-1}. Using the baseline corrected data, the amide I absorbance band at 1650 cm^{-1} was measured. Any pixels with an amide I absorbance outside the range of 0.4 to 1.0 absorbance units were removed from the images. No pixel co-addition or outlier removal was performed. A boxcar smoothing algorithm was then applied to the raw (non-baseline corrected) data using eleven smoothing points.

As far as feature selection is concerned, no mean centering was performed. Each pixel within the image was subjected to a normalization (2-Norm) procedure. Following the normalization, each individual spectrum was converted to the first derivative. There was no integration of the pixel data.

For feature extraction, the first seven principal components were calculated, using the weights calculated from the original training set. Once the principal components were calculated, the distances to the centers of the two clusters found in the training set were calculated on a pixel-by-pixel basis and used to determine the Gram-classification. The

squared Euclidean distance metric was used as it was the metric applied in conjunction with the K-means clustering algorithm within the genetic algorithm.

3.4.1 Validation Test 1

Each of the 19 specimens included in the training set had been cultured three times, yielding a total of 57 samples for which infrared images were acquired. As shown in Table 3-1, the training set included duplicate samples of 5 of the 19 specimens. For validation of the Gram classification model developed using this training set, 19 other samples among the set of 57 samples were selected. These samples are tabulated in Table 3-8.

Table 3-8: Validation Test 1 – Bacterial Species and Specimens

Strain Name	Specimen #	Training Set Culture(s) #	Validation Set Culture #	Gram-
Klebsiella oxytoca	112	2	1	Negative
Hafnia alvei	115	2	1	Negative
Klebsiella pneumoniae	145	2	3	Negative
Klebsiella pneumoniae	165	4	2	Negative
Escherichia coli	1125-26	2	4	Negative
Escherichia coli	1156-2	3	1	Negative
Salmonella berta	804	2	4	Negative
Salmonella derby	4359	1, 2	3	Negative
Escherichia coli 0157:H7	149	1	2	Negative
Salmonella heidelberg	3221	1, 2	3	Negative
Streptococcus xylosus	244	2, 3	1	Positive
Clostridium sporogenes	241	1	2	Positive
Listeria monocytogenes	4404	1	2	Positive
Listeria monocytogenes	4410	1, 2	3	Positive
Listeria monocytogenes	4426	1	2	Positive
Listeria monocytogenes	4749	2	3	Positive
Staphylococcus aureus	251	4	3	Positive
Listeria monocytogenes	688	3	2	Positive
Listeria monocytogenes	1116-2	1, 2	3	Positive

The infrared images acquired for these samples were subjected to the pre-processing variables selected by the genetic algorithm. Following the removal of the

pixels outside of the acceptable amide I range of 0.4 to 1.0 absorbance units, 6,037 of the original 19,456 pixels remained for classification purposes. Figure 3-13 illustrates the remaining pixels in the image for the Escherichia coli sample after the amide I absorbance tolerance was evaluated.

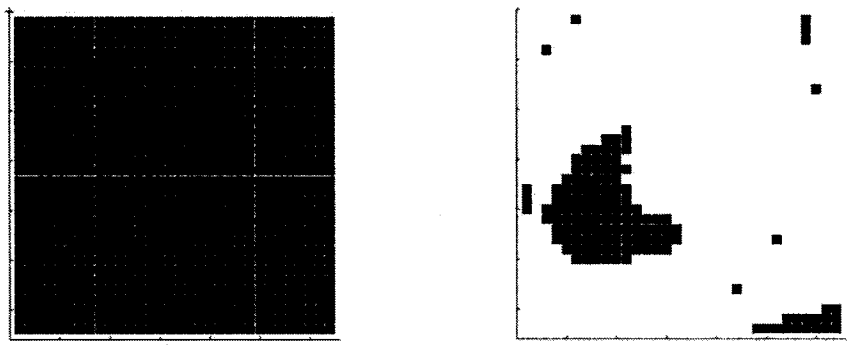

Figure 3-13: Image of Escherichia coli Sample: Pixel Utilization Before and After Selection Based on Amide I Tolerance

Boxcar smoothing was then applied with 11 points followed by normalization and the first derivative. Following the initial pre-processing treatments, the seven principal component weights from the sub-optimal genetic algorithm solution were applied to each individual spectrum. The principal components were projected onto a three dimensional plane and superimposed on the original clustering data resulting from the genetic algorithm. Although it is difficult to visualize the complete dimensionality of the data, it is apparent in Figure 3-14 that the validation data falls within the boundaries of the data for the training set. The image centers of the validation data are represented by the o's in the image while the +'s represent the centers of the original training set clusters.

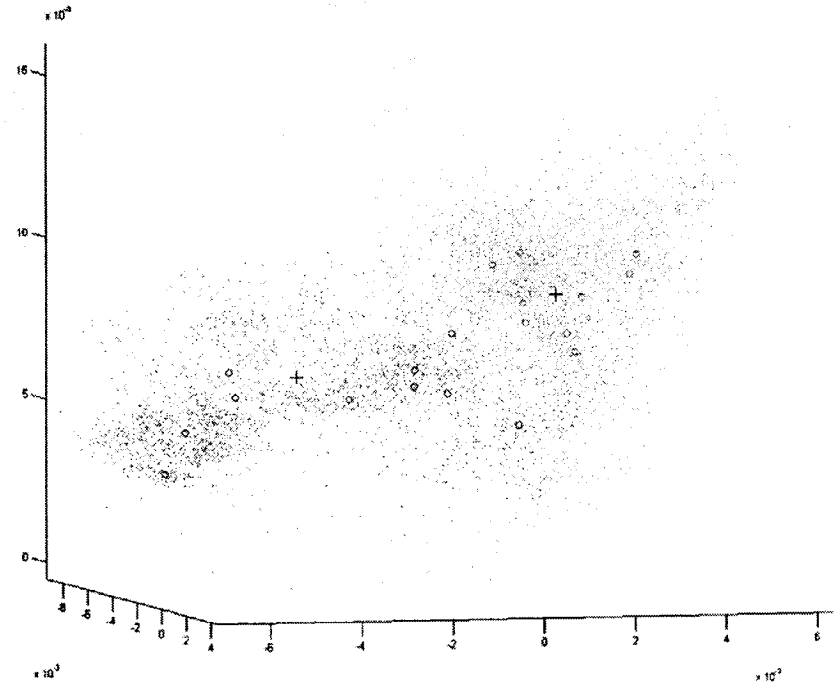

Figure 3-14: Validation Test 1 – Cluster Plot

In order to determine the classification of the samples in the validation set, the distances between the centers of each image and the centroids of the original clusters produced by the genetic algorithm were measured and the sample was classified based on the shortest center-to-centroid distance. This procedure resulted in 100% accurate classification of the samples in the validation data set as Gram-positive versus Gram-negative.

In order to determine the confidence level of the classifications, the spectral distance to the centroids of the Gram positive and Gram negative clusters was measured for each pixel in each of the images of the validation set. The percent confidence was calculated as the number of correctly classified pixels, based on closer proximity to the Gram positive or the Gram negative cluster, divided by the final number of retained pixels in the image as illustrated in Table 3-9.

Table 3-9: Results of Validation Test 1

Strain Name	Culture #	Final # Pixels	# Pixels Correct	% Confidence	Gram-	Distance to Cluster
Klebsiella oxytoca	1	463	463	100%	Negative	1.8491×10^{-5}
Hafnia alvei	1	94	94	100%	Negative	2.5750×10^{-5}
Klebsiella pneumoniae	3	301	301	100%	Negative	1.0883×10^{-5}
Shigella flexneri	2	401	284	70.8%	Negative	3.3588×10^{-5}
Escherichia coli	4	154	107	69.5%	Negative	2.7111×10^{-5}
Escherichia coli	1	172	171	99.4%	Negative	2.1244×10^{-5}
Salmonella berta	4	26	21	80.8%	Negative	5.3104×10^{-5}
Salmonella derby	3	109	96	88.1%	Negative	2.9155×10^{-5}
Escherichia coli 0157:H7	2	136	134	98.5%	Negative	5.7953×10^{-6}
Salmonella heidelberg	3	585	585	100%	Negative	1.3862×10^{-5}
Streptococcus xylosus	1	369	369	100%	Positive	1.5291×10^{-5}
Clostridium sporogenes	2	484	484	100%	Positive	2.0286×10^{-5}
Listeria monocytogenes	2	312	312	100%	Positive	4.1102×10^{-5}
Listeria monocytogenes	3	430	430	100%	Positive	2.7379×10^{-5}
Listeria monocytogenes	2	487	487	100%	Positive	2.0458×10^{-5}
Listeria monocytogenes	3	393	392	99.7%	Positive	3.3254×10^{-5}
Staphylococcus aureus	3	549	549	100%	Positive	1.3731×10^{-5}
Listeria monocytogenes	2	305	305	100%	Positive	1.4888×10^{-5}
Listeria monocytogenes	3	267	267	100%	Positive	9.4104×10^{-5}

In order to evaluate another form of goodness of fit, the centroid location of each individual image was measured in relation to the centers of the clusters formed by the genetic algorithm. The measurements were carried out using the squared Euclidean distance metric and compared to the radius of the original cluster formed by the genetic algorithm. The radius of the Gram-positive cluster was 1.6880×10^{-4}, and the radius of the Gram-negative cluster measured 1.0819×10^{-4}. Table 3-9 illustrates the measures of each of the Gram-positive and Gram-negative images to the centroids. It is apparent that each of the validation images fell within the radius boundaries of the training set.

Examination of both the pixel to pixel clustering accuracy and the measure of the image centroids to each of the cluster centers indicates that this clustering sub-optimum solution is an effective classification tool. Each of the validation samples fell within the bounds of the clustering model and the confidence levels were acceptable.

3.4.2 Validation Test 2

The results obtained in the first validation test show that the model developed for Gram classification could be successfully applied to additional samples of the specimens included in the training set. In the second validation test, the performance of the model in regards to specimens not included in the training set was assessed.

To select samples for the second validation test, the bacterial spectral library was filtered to remove all replicates of the specimens used in the training set. Six Gram-positive and six Gram negative species were then randomly selected. The spectral images were processed and classified based on the clustering model determined by the genetic algorithm in the same manner as described above and illustrated in Figure 3-3. Following the removal of the pixels outside of the acceptable amide I range of 0.4 to 1.0 absorbance units, the total number of pixels was reduced from 12,288 to 3,379 conserving 27.5% of the original validation data. In the case of two of the samples, less than 10 pixels were retained, indicating that the deposition of the sample film on the slide was inadequate. Accordingly, these samples were excluded from the data analysis. Table 3-10 illustrates the classification accuracy for each image on a pixel-by-pixel basis and a confidence value. Table 3-10 also illustrates the distance of the center of each image to the corresponding cluster centroids of the training set. All of these distances fall well within the boundary values of the classification model.

Table 3-10: Results of Validation Test 2

Strain Name	Culture #	Final # Pixels	# Pixels Correct	% Confidence	Gram-	Distance to Cluster
Aeromonas hydrophila	1	465	465	100%	Negative	4.9466×10^{-5}
Aeromonas hydrophila	1	148	134	90.5%	Negative	2.7659×10^{-5}
Salmonella typhmurium	2	400	385	96.3%	Negative	9.1221×10^{-6}
Salmonella typhmurium	3	212	199	93.9%	Negative	1.3910×10^{-5}
Vibrio parahaemolyticus	1	379	352	92.9%	Negative	2.6723×10^{-5}
Escherichia coli 8739	2	421	420	99.7%	Negative	1.6974×10^{-5}
Listeria ivanovii	1	9	-	-	-	-
Listeria murrayi	3	8	-	-	-	-
Clostridia perfringens	1	451	451	100%	Positive	3.1181×10^{-5}
Listeria monocytogenes	3	176	176	100%	Positive	2.5636×10^{-5}
Listeria monocytogenes	1	324	324	100%	Positive	3.2763×10^{-5}
Listeria monocytogenes	1	386	386	100%	Positive	1.5566×10^{-5}

Examination of the pixel to pixel clustering accuracy, as well as the measure of the validation image centers to each of the cluster centers, indicates that the clustering model is an effective classification tool for specimens not included in the training set. Each of the validation samples fell within the bounds of the clustering model and the confidence levels were more than acceptable.

4 Conclusion

The primary aim of this study was to examine the utilization of a genetic algorithm to optimize the pre-processing of infrared image data, with the secondary aim being to conserve as much of the original pixel data as possible. The data set selected for this purpose consisted of infrared images of bacterial cultures, and the classification task investigated was the discrimination between Gram-positive and Gram-negative bacteria. The use of the genetic algorithm was explored with a training set consisting of 12 Gram-positive and 12 Gram-negative specimens. The genetic algorithm evaluated combinations of variables pertaining to bacterial film thickness tolerances, baseline correction, pixel co-addition, outlier removal, smoothing, mean centering, normalization, derivatization, integration and principal component selection and employed a fitness function that utilized a score incorporating the classification accuracy (assigned a weight of 80%) and the number of remaining pixels after all of the pre-processing was accomplished (assigned a weight of 20%) When the genetic algorithm was applied to the infrared image data for the samples in the training set, the clustering of the infrared images on a pixel-by-pixel basis yielded a classification accuracy of approximately 97.5%; the corresponding value for classification on an image-by-image basis was 100%. With respect to the secondary aim of the algorithm, the proportion of the pixel data retained from the original images was 28.6%.

Applying the genetic algorithm to the spectral data for the training set yielded an appropriate combination of pre-processing variables for clustering of Gram-positive and Gram-negative specimens. To test the robustness of this combination of pre-processing

variables, two validation tests were performed, the first using replicate images of the specimens included in the training set and the second using images of a different set of specimens. Following pre-processing of the data in accordance with the procedures established for the training set with the use of the genetic algorithm, the validation samples were classified based on the squared Euclidean distances to the centroids of the Gram-positive and Gram-negative clusters of the training set. All the validation samples were classified correctly on an image-by-image basis and with a fairly high accuracy on a pixel-by-pixel basis. Furthermore, as summarized in Table 4-1: Comparison of Distances to Clusters, the average distances of the Gram-positive and the Gram-negative samples in both the first and second validation sets to the centroids of the corresponding clusters were well within the bounds for the training set.

Table 4-1: Comparison of Distances to Clusters

	Gram-Negative	Gram-Positive
Training Set Cluster Radius	1.0819×10^{-4}	1.6880×10^{-4}
Validation Set 1 – Avg. Dist. To Cluster	2.3898×10^{-5}	3.1166×10^{-5}
Validation Set 2 – Avg. Dist. To Cluster	2.3976×10^{-5}	2.6287×10^{-5}

The validated success of the initial pass of the genetic algorithm indicates that it is an effective time-saving tool for the optimization of pre-processing variables for clustering and classification tasks. As discussed in Chapter 3, based on the results obtained in this work, a second pass could be attempted with several variables removed or replaced in order to increase the speed and accuracy of the algorithm. For instance, the baseline or raw data flag as well as the co-addition modules could be removed, significantly reducing the computational expense of the algorithm by five bits. The spectral data could also be limited to the region between 1817 cm^{-1} and 1053 cm^{-1}, thereby reducing the size of the data set by approximately one-third and again reducing

the computational expense of the algorithm. Alternative approaches for clustering of the spectral data could also be investigated. Applying a "learning" algorithm such as a neural network or k-nearest neighbors algorithm would most probably be highly effective. However, it is important to note that learning algorithms tend to overfit the data and may not produce correct classifications for samples not included in the training set.

In conclusion, as advances in infrared imaging technology result in increasingly large sets of spectral image data, researchers require improved means of handling and interpreting data and applying this technology to perform particular analytical tasks. It is the appropriate combination of data acquisition, processing and analysis techniques that make it possible to effectively classify a hyperspectral data set. The research presented in this thesis demonstrates the effectiveness of a genetic algorithm as a tool for selecting pre-processing variables for a given classification task. The use of a genetic algorithm allows for the sub-optimization of the pre-processing variables without the intrinsic trial and error reasoning of the researcher. Using the genetic algorithm thus releases human resources, allowing them to be allocated to other tasks while minimizing the influence of human error. The sub-optimum solution was produced by the genetic algorithm in a reasonable time frame of less than seven hours, as opposed to manually optimizing the combination of variables over an extended period of time, and modifications of the initial set of variables, as discussed above, could further reduce the computational time. Finally, the genetic algorithm not only helps in the development of an unsupervised sub-optimum solution, but can also enhance the understanding of the relationships between pre-processing variables and of their effects on overall analytical performance.

5 References

Adams, M.J. (1995). *Chemometrics in Analytical Spectroscopy.* Cambridge: The Royal Society of Chemistry.

Beebe, Kenneth R.; Pell, Randy J.; Seasholtz, Mary Beth. (1998). *Chemometrics: A Practical Guide.* New York: Wiley Publications.

Bourke, P. (1993). *Discrete Fourier Transform.* Victoria, Australia: Swinburne University of Technology http://astronomy.swin.edu.au/~pbourke/other/dft/

Burns, D.H. (2001). *Chemometrics: Analysis of Chemical Data (Chemistry 567A Lecture Handouts.)* Montreal: McGill University.

Davies, A.; Board, R. (1998). *The Microbiology of Meat and Poultry.* London, UK: Blackie Academic & Professional.

Davis, Sumner P.; Abrams, Mark C.; Brault, James W. (2001). *Fourier Transform Spectrometry.* San Diego, USA: Academic Press.

Firkin, B.G.; Whitworth, J.A. (1987). *Dictionary of Medical Eponyms.* Nashville, USA: Parthenon Publishing.

Gilbert, R.J.; Goodacre, R.; Woodward, A.M.; Kell, D.B. (1997). *Genetic Programming.* Aberystwyth, UK: Institute of Biological Sciences: Analytical Chemistry

Hashimoto, T.; Birch, W.X. (1996). *Gram Stain.* Chicago, USA: Loyola University.

Jarvis, Roger M.; Goodacre, Royston. (2004). *Genetic Algorithm Optimization for Pre-Processing and Variable Selection of Spectroscopic Data.* Manchester, UK: Oxford University Press.

Kirkwood, Jonah; Al-Khaldi, Sufian F.; Mossoba, Magdi M.; Sedman, Jacqueline; Ismail, Ashraf A. (2004). *Fourier transform infrared bacteria identification with the use of a focal-plane-array detector and microarray printing.* Montreal, Canada: Applied Spectroscopy 58(11)

Kowaski; Illman; Sharaf. (1986). *Chemometrics.* New York: Wiley Publications.

MacKay, D.J.C. (2003). *Information Theory, Inference, and Learning Algorithms.* Cambridge: Cambridge University Press.

Matlab R13, Statistics Toolbox. (2002). Boston: Mathworks.

Naumann, D. (2000). *Infrared spectroscopy in microbiology. In: Encyclopedia of Analytical Chemistry.* New York: Meyers, R.A., Ed. John Wiley & Sons

Rainieri, S.; Pagliarini, G. (2001). *Data Processing Technique Applied to the Calibration of a High Performance FPA Infrared Camera.* Parama, Italy: Elsevier Science B.V.

Russell, S.; Norvig, P. (1995). *Artificial Intelligence A modern Approach.* New Jersey, USA: Prentice-Hall Inc.

UOT (1995). *Introduction to Clinical Microbiology.* Texas, USA: University of Texas – Houston Medical School http://medic.med.uth.tmc.edu.

Van Den Broek, W.H.A.M.; Wienke, D.; Melssen, W.J.; Buydens, L.M.C. (1997). *Optimal Wavelength Range Selection by a Genetic Algorithm for Discrimination Purposes is Spectroscopic Infrared Imaging.* Nijmegen, Netherlands: Society of Applied Spectroscopy

Wolsky, A.M; Daniels, E.J.; Giese, R.F.; Harkness, J.B.L.; Johnson, L.R.; Rote, D.M.; Zwick, S.A. (1989). *Applied Superconductivity.* New Jersey, USA: Noyes Data Corporation.

6 Appendix: Genetic Algorithm Matlab Code

6.1 Main Genetic Function

```
% Tom Pinchuk
% Genetic Algorithm
% Revision September 2005

% Bit layout -  27 Bits representing numbers 0 to 134,217,727
%               Combinations.
%               See "GA Flow Chart.ppt" for more information
%
% Input Arguments:
% imageFile:    The filename of the image without an extension.
% numIterations:The number of iterations to compute when running the
%               genetic algorithm. This number must be an even number.

function [iData, oData]=A00_runGenetic(imageFile, numIterations)
format long e;

% Check for errors.
if rem(numIterations,2)~=0
    disp('Number of iterations must be an even number.');
    quit;
end

% Load the image Data and the output data if it is available. If not set
% the number of iterations to 0.
disp('Loading the data.');
[iData]=A01_LoadData(imageFile);
outputFile=strcat(imageFile, 'OUT');
bOdd=0;
if exist(strcat(outputFile,'.mat'),'file') == 0
    disp('Creating Output File.');
    curIteration=0;
    bOdd=1;
else
    disp('Loading Output File.');
    load(outputFile);
    curIteration=length(oData.DNA);
    if rem(curIteration,2)==0
        bOdd=1;
    end
end

for cntIter=(curIteration+1):(curIteration+numIterations)
    disp(['   Iteration: ' num2str(cntIter) ' of '...
        num2str(curIteration+numIterations)]);

    % Reproduce and get two new DNA strands only when the iterations are
    % odd. Therefore we always finish with an even number.
    if cntIter==1
        DNA=[134217728,0];
        oData.DNA(cntIter)=DNA(1);
    elseif rem(cntIter,2)==bOdd
        [DNA, bOdd, oData]=reproduce(cntIter, oData, bOdd);
        oData.DNA(cntIter)=DNA(1);
    else
        oData.DNA(cntIter)=DNA(2);
    end
    disp([' Digital DNA: ' num2str(oData.DNA(cntIter))]);

    % Split up the DNA into the correct functions. Original DNA contains
    % 27 bits
    DNA_PCA = bitshift(oData.DNA(cntIter),-23);
    tmpDNA = oData.DNA(cntIter) - bitshift(DNA_PCA,23);
    DNA_Feature = bitshift(tmpDNA,-17);
    tmpDNA = tmpDNA - bitshift(DNA_Feature,17);
```

```
DNA_Smooth = bitshift(tmpDNA,-13);
tmpDNA = tmpDNA - bitshift(DNA_Smooth,13);
DNA_Coadd = bitshift(tmpDNA,-11);
tmpDNA = tmpDNA - bitshift(DNA_Coadd,11);
DNA_Outlier = bitshift(tmpDNA,-9);
tmpDNA = tmpDNA - bitshift(DNA_Outlier,9);
DNA_Base = bitshift(tmpDNA,-8);
DNA_Amide = tmpDNA - bitshift(DNA_Base,8);

% Check the Amide 1 tollerances and update the output data.
[rData]=A02_RemoveAmide(iData,DNA_Amide);
oData.PctPixel(cntIter,1)=rData.NumPixels/iData.NumPixels;
oData.PctImage(cntIter,1)=rData.NumImages/iData.NumImages;
if (rData.NumPixels==0) || (rData.NumImages<iData.NumImages)
    oData.Score(cntIter,1:3)=0;
    oData.totScore(cntIter)=0;
end
disp(['      -> Pixels: ' num2str(rData.NumPixels) ' / Images: '...
    num2str(rData.NumImages)]);

if (rData.NumPixels>2) && (rData.NumImages==iData.NumImages)
    % Determine to use baseline corrected or raw data
    [rData] = A03_UseRaw(iData, rData, DNA_Base);
    % New reversing functions - coadd images
    [rData] = A05_CoaddImages(iData, rData, DNA_Coadd);
    disp(['      -> Pixels: ' num2str(rData.NumPixels) ' / Images: '...
        num2str(rData.NumImages)]);
end
oData.PctPixel(cntIter,2)=rData.NumPixels/iData.NumPixels;
oData.PctImage(cntIter,2)=rData.NumImages/iData.NumImages;
if (rData.NumPixels==0) || (rData.NumImages<iData.NumImages)
    oData.Score(cntIter,1:3)=0;
    oData.totScore(cntIter)=0;
end

% Remove Outliers - use 04b When reversed with coadd algorithm
if (rData.NumPixels>2) && (rData.NumImages==iData.NumImages)
    [rData] = A04b_RemoveOutliers(iData, rData, DNA_Outlier);
    disp(['      -> Pixels: ' num2str(rData.NumPixels) ' / Images: '...
        num2str(rData.NumImages)]);
end
oData.PctPixel(cntIter,3)=rData.NumPixels/iData.NumPixels;
oData.PctImage(cntIter,3)=rData.NumImages/iData.NumImages;
if (rData.NumPixels==0) || (rData.NumImages<iData.NumImages)
    oData.Score(cntIter,1:3)=0;
    oData.totScore(cntIter)=0;
end

% Feature Selection / Combine data / PCA / Cluster (need at least 3
% pixels or it will not create a proper linkage or cluster.
if (rData.NumPixels>2) && (rData.NumImages==iData.NumImages)
    [rData] = A06_SmoothImages(iData, rData, DNA_Smooth);
    [rData] = A07_FeatureSelection(iData, rData, DNA_Feature);
    [rData] = A08_CombineImageData(iData, rData);
    [rData, pc] = A09_PCA(iData, rData, DNA_PCA);
    % Adjusted for kMeans
    [rData] = A10b_ClusterFunction(rData);
    clear pc;
    % Determine Scores
    % 80% Accuaracy    (1)
    % 00% Images       (2)
    % 20% Data Points  (3)
    % New... oData.totScore(Num)
    oData.Score(cntIter,1)=rData.Score;
    oData.Score(cntIter,2)=oData.PctImage(cntIter,3);
    oData.Score(cntIter,3)=rData.NumPixels;
    oData.totScore(cntIter)=...
        (oData.Score(cntIter,1)*.8)+...
        (oData.PctImage(cntIter,3)*0)+...
        ((oData.Score(cntIter,3)/iData.NumPixels)*.2);
else
```

```
        oData.Score(cntIter,1:3)=0;
        oData.totScore(cntIter)=0;
    end
    disp(['-> Scores: ' num2str(oData.totScore(cntIter)*100) ' = '...
        num2str(oData.Score(cntIter,1)*100) '% and '...
        num2str(oData.Score(cntIter,2)*100) '% Images and '...
        num2str(oData.Score(cntIter,3)) ' Pixels.']);
    tmpOutFile=strcat('__',outputFile,num2str(cntIter));
    save(tmpOutFile,'oData');
end
save(outputFile,'oData');
```

6.2 Data Loading Function

```
%
% Tom Pinchuk
%
% Revised for "Rescue Pixels" 9/25/2005
% Redesigned only to use the sample data provided. Support for additional
% file types has been temporarily removed. New input file type is a CSV.
% Preloaded files will just be retrieved using the .mat file.
%
% Assumptions:
% - All images have the same characteristics and resolutions. Only need 1
%   ENVI file coorinating with the first file name to load all of the
%   images.
% - The bands are in numerical order from low to high (we can verify in the
%   ENVI header file.
%
%
% Input Arguments:
% fileName:     The name of the input data set - no need for a file
%               extension.

function [iData] = A01_LoadData(fileName)

% Constants (Per flow chart - no longer read from text file)
cstBasePt(1)=980;
cstBasePt(2)=1780;
cstRespBand=1650;
cstAmideRange(1)=0.4;
cstAmideRange(2)=1.4;

% Creating file names to search for.
fileCSV=strcat(fileName,'.csv');
fileMAT=strcat(fileName,'.mat');

% Check if file has been previously loaded - if so load information from
% the mat file. If not proceed to building the initial data structure.
if exist(fileMAT,'file') >= 1
    load(fileMAT);
else
    % Load the text file containing the images. The extension is .inf
    [fileContents]=textread(fileCSV,'%s','delimiter',',');
    lengthTextFile=length(fileContents);
    iData.NumImages=lengthTextFile/2;    %Set the number of images property

    % Read the ENVI file.
    %tmpFile=fileContents(1);
    %fileHDR=char(tmpFile(1:length(tmpFile)-4));
    [enviInfo]=textread('000.hdr','%s');
    iData.NumCols=str2num(enviInfo{10,1});  %Set the number of Columns
    iData.NumRows=str2num(enviInfo{13,1});  %Set the number of Rows
    %Set the number of bands - number of data points for each pixel
    iData.NumDataPoints=str2num(enviInfo{16,1});
    %Set the number of Classes
   iData.NumClasses=length(unique(fileContents(2:2:length(fileContents))));

    datatype=str2num(enviInfo{29,1});
```

```
switch datatype
    case {1}
        datastring='int8';
    case {2}
        datastring='int';
    case {3}
        datastring='int64';
    case {4}
        datastring='float';
    case {5}
        datastring='double';
    case {6}
        error('readenvi.m file cannot handle complex data...')
end
for cntBand=0:(iData.NumDataPoints-1)   %Set the individual bands
    iData.Wavenumbers(cntBand+1) = str2num(enviInfo{(49+cntBand),1});
end
offset=str2num(enviInfo{20,1});

% Load the raw image data
for cntImages=1:iData.NumImages
    indexFName = 2*cntImages-1;
    indexFClass = 2*cntImages;
    tmpFName=char(fileContents(indexFName));
    fid=fopen(tmpFName,'r');
    [datInfo]=fread(fid,offset,'int8');
    [spectra]=fread(fid,datastring);
    specimg=reshape...
        (spectra,[iData.NumRows,iData.NumCols,iData.NumDataPoints]);
    for i=1:iData.NumDataPoints
        specimg(:,:,i)=specimg(:,:,i)';
    end
    iData.Image(cntImages).rawData=specimg;
    fclose(fid);
    iData.Image(cntImages).class=...
        str2num(char(fileContents(indexFClass)));
end

% Calculate additional constants
iData.NumPixels=iData.NumImages*iData.NumCols*iData.NumRows;
iData.Resolution=(iData.Wavenumbers(11)-iData.Wavenumbers(1))/10;

% Baseline Correct each image and determine response - fill in image
% map - determine to use image or not
base(1)=round((cstBasePt(1)-iData.Wavenumbers(1))/iData.Resolution+1);
base(2)=round((cstBasePt(2)-iData.Wavenumbers(1))/iData.Resolution+1);
response=round((cstRespBand-iData.Wavenumbers(1))/iData.Resolution+1);
tmpRun=base(2)-base(1);
for cntImages=1:iData.NumImages
    iData.Image(cntImages).numGoodPixels=0;
    for cntRow=1:iData.NumRows
        for cntCol=1:iData.NumCols
            absorbance(1)=...
                iData.Image(cntImages).rawData(cntRow,cntCol,base(1));
            absorbance(2)=...
                iData.Image(cntImages).rawData(cntRow,cntCol,base(2));
            tmpSlope=(absorbance(2)-absorbance(1))/tmpRun;
            tmpConst=absorbance(2)-(tmpSlope*base(2));
            for cntPT=1:iData.NumDataPoints
                iData.Image(cntImages).baseData(cntRow,cntCol,cntPT)=...
                    iData.Image(cntImages).rawData(cntRow,cntCol,cntPT)...
                    - (tmpSlope*iData.Wavenumbers(cntPT)+tmpConst);
            end
            iData.Image(cntImages).AmideResponse(cntRow,cntCol)=...
                iData.Image(cntImages).baseData(cntRow,cntCol,response);
            if (iData.Image(cntImages).AmideResponse(cntRow,cntCol)...
                    <cstAmideRange(1)) ||...
                    (iData.Image(cntImages).AmideResponse(cntRow,cntCol)>...
                    cstAmideRange(2))
                iData.Image(cntImages).pixlMap(cntRow,cntCol)=0;
            else
```

```
                    iData.Image(cntImages).pixlMap(cntRow,cntCol)=1;
                    iData.Image(cntImages).numGoodPixels=...
                        iData.Image(cntImages).numGoodPixels+1;
                end
            end % Count Columns
        end % Count Rows
        if iData.Image(cntImages).numGoodPixels==0
            iData.Image(cntImages).useImage=0;
        else
            iData.Image(cntImages).useImage=1;
        end
    end % Count Images

    % Save the file
    save(fileMAT,'iData');
end
```

6.3 Amide I Removal Function

```
%
% Tom Pinchuk
%
% 9/30/2005
% New process to optimize the range of acceptable Amide 1 absorbance
% responces from 0.4 to 1.4. The increments are 6/90. The range is
% determined by two sets of four bits each representing a range point.
%
% Assumptions:
%
%
% Input Arguments:
% iData:        The initial specatral data as determined by the load data
%               algorithm.
% DNA:          The 8-bit DNA strand representing both of the Amide 1
%               tolerances. If they are the same, the scores are zero. This
%               must be checked in the main GA script.
%               Bits 8-5: Numerical value * 6/90 + 0.4 = Range Pt 1
%               Bits 4-1: Numerical value * 6/90 + 0.4 = Range Pt 2
%
%
% Output Arguments:
% rData:        The data used during processing of the spectra.

function [rData] = A02_RemoveAmide(iData, DNA)

% Constants (Per flow chart - no longer read from text file)
cstRespBand=1650;
cstAmideRange(1)=0.4;
cstAmideRange(2)=1.4;

% Determine Ranges - and sort if necessary
tmpRange(1)=bitshift(DNA,-4);
tmpRange(2)=DNA - bitshift(tmpRange(1),4);
tmpRange(1)=double(tmpRange(1))*6/90+cstAmideRange(1);
tmpRange(2)=double(tmpRange(2))*6/90+cstAmideRange(1);
range=sort(tmpRange);
disp(['    - Amide 1 Range: ' num2str(range(1)) ' to ' num2str(range(2))]);

% Ensure that the range values are not identical and check the amide
% responses.
rData.NumPixels=0;
rData.NumImages=0;
if range(1)~=range(2)
    for cntImages=1:iData.NumImages
        rData.Image(cntImages).numGoodPixels=0;
        for cntRow=1:iData.NumRows
            for cntCol=1:iData.NumCols
                if (iData.Image(cntImages).AmideResponse(cntRow,cntCol)<range(1)) ||...
                        (iData.Image(cntImages).AmideResponse(cntRow,cntCol)>range(2))
```

```
                    rData.Image(cntImages).pixlMap(cntRow,cntCol)=0;
                else
                    rData.Image(cntImages).pixlMap(cntRow,cntCol)=1;

rData.Image(cntImages).numGoodPixels=rData.Image(cntImages).numGoodPixels+1;
                    rData.NumPixels=rData.NumPixels + 1;
                end
            end % Count Columns
        end % Count Rows
        if rData.Image(cntImages).numGoodPixels==0
            rData.Image(cntImages).useImage=0;
        else
            rData.Image(cntImages).useImage=1;
            rData.NumImages=rData.NumImages+1;
        end
    end % Count Images

    % Update the rest of the running data structure. - Do not include the
    % data parameter.
else
    rData.NumPixels=0;
    rData.NumImages=0;
end
```

6.4 Baseline or Raw Selection Function

```
%
% Tom Pinchuk
%
% 10/02/2005
% New process to determine if the RAW data or the baseline corrected data
% should be utilized for future computations.
%
% This function also zeros out the data values of any pixels not in use in
% the rData structure.
%
% Assumptions:
%
%
% Input Arguments:
% iData:        The initial specatral data as determined by the load data
%               algorithm.
% rData:        The running Data up until now.
% DNA:          The 1-bit DNA strand representing if the Raw data or baseline
%               corrected data should be used.
%               0: Keep the raw data
%               1: Use the baseline corrected data
%
% Output Arguments:
% rData:        The data used during processing of the spectra.

function [rData] = A03_UseRaw(iData, rData, DNA)
if DNA==0
    disp(['    - Using RAW Data']);
else
    disp(['    - Using Baseline Corrected Data']);
end
for cntImages=1:iData.NumImages
    if DNA==0
        rData.Image(cntImages).data = iData.Image(cntImages).rawData;
    else
        rData.Image(cntImages).data = iData.Image(cntImages).baseData;
    end
    for cntRow=1:iData.NumRows
        for cntCol=1:iData.NumCols
            if rData.Image(cntImages).pixlMap(cntRow,cntCol)==0
                rData.Image(cntImages).data(cntRow,cntCol,1:iData.NumDataPoints)=0;
            end
        end % Counting Columns
```

```
    end % Counting Rows
end
```

6.5 Pixel Co-Addition Function

```
%
% Tom Pinchuk
%
% Modified 10/02/2005
%
% This function is used to coadd the data in the images.
%
% Images are coadded in adjacent groups of 4, 16 and 32 pixels. The input
% is a 2-bit binary referencing the coadd values. If nothing is so be
% coadded, than the images are combined for further processing.
%
% Assumptions:
%
%
% Input Arguments:
% iData:        The initial specatral data as determined by the load data
%               algorithm.
% rData:        The running Data up until now.
% DNA:          The 2-bit DNA strand representing the acceptable number of
%               pixels to coadd.
%               00: Keep the entire image
%               01: Coadd 4 adjacent pixels  [2x2]
%               10: Coadd 16 adjacent pixels [4x4]
%               11: Coadd 64 adjacent pixels [8x8]
%
% Output Arguments:
% rData:        The data used during processing of the spectra.
function [rData] = A05_CoaddImages(iData, rData, DNA)
% Compute the square root of the number of pixels to coadd.
switch DNA
    case {0}
        CoaddLimit=0;
    case {1}
        CoaddLimit=2;
    case {2}
        CoaddLimit=4;
    case {3}
        CoaddLimit=8;
end
disp(['    - Coadding: ' num2str(CoaddLimit) 'x' num2str(CoaddLimit) ' images']);
if CoaddLimit==0
    for cntImage=1:iData.NumImages
        cntPixel=0;
        if rData.Image(cntImage).useImage==1
            rrData.CmbImg(cntImage).useImage=1;
            for cntRow=1:iData.NumRows
                for cntCol=1:iData.NumCols
                    if rData.Image(cntImage).pixlMap(cntRow, cntCol)==1
                        cntPixel=cntPixel+1;

rrData.CmbImg(cntImage).data(cntPixel,:)=rData.Image(cntImage).data(cntRow,cntCol,:);
                    end
                end % Count Column
            end % Count Row
            rrData.CmbImg(cntImage).numPixels=cntPixel;
        else
            rrData.CmbImg(cntImage).useImage=0;
            rrData.CmbImg(cntImage).numPixels=0;
        end
    end % Count Images
else
    % Start from the top left looking for adjacent good pixels. With each
    % iteration - update the pixel map and record the coadded image.
     for cntImage=1:iData.NumImages
```

```
        cntPixel=0;
        if rData.Image(cntImage).useImage==1
            for cntRow=1:(iData.NumRows-CoaddLimit+1)
                for cntCol=1:(iData.NumCols-CoaddLimit+1)
                    if rData.Image(cntImage).pixlMap(cntRow, cntCol)==1
                        %Check to see if the adjacent points are all good
                        boolAllGood=1;
                        for cntDR=cntRow:(cntRow+CoaddLimit-1)
                            for cntDC=cntCol:(cntCol+CoaddLimit-1)
                                if rData.Image(cntImage).pixlMap(cntDR,cntDC)==0,
boolAllGood=0; end
                            end % dbl Count Columns
                        end % dbl Count Rows
                        %If they are all good - coadd them and the remove
                        %their pixel flags
                        if boolAllGood==1
                            cntPixel=cntPixel+1;
                            rrData.CmbImg(cntImage).data(cntPixel,:)=...

sum(sum(rData.Image(cntImage).data(cntRow:(cntRow+CoaddLimit-
1),cntCol:(cntCol+CoaddLimit-1),:)));
                            rData.Image(cntImage).pixlMap(cntRow:(cntRow+CoaddLimit-
1),cntCol:(cntCol+CoaddLimit-1))=0;
                        end
                    end
                end % Count Column
            end % Count Row
            if cntPixel>0
                rrData.CmbImg(cntImage).useImage=1;
            else
                rrData.CmbImg(cntImage).useImage=0;
            end
            rrData.CmbImg(cntImage).numPixels=cntPixel;
        else
            rrData.CmbImg(cntImage).useImage=0;
            rrData.CmbImg(cntImage).numPixels=0;
        end
    end % Count Images
end

%Update the total number of images and pixels.
rrData.NumPixels=0;
rrData.NumImages=0;
rrData.NumDataPoints=iData.NumDataPoints;
rrData.Wavenumbers=iData.Wavenumbers;
for cntImage=1:iData.NumImages
    if rrData.CmbImg(cntImage).useImage==1
        rrData.NumImages=rrData.NumImages+1;
        rrData.NumPixels=rrData.NumPixels+rrData.CmbImg(cntImage).numPixels;
    end
end
rData=rrData;
```

6.6 Outlier Removal Function

```
%
% Tom Pinchuk
%
% Modified 10/02/2005
% This file is used to determine and remove outliers. The input is a 2 bit
% binary value that references removing outliers beyond certain standard
% deviations.
%
% B Version is when Coadding and outliers are reversed
%
% The image maps and image flags will be updated with this function.
%
% Assumptions:
%
```

```
%
% Input Arguments:
% iData:        The initial specatral data as determined by the load data
%               algorithm.
% rData:        The running Data up until now.
% DNA:          The 2-bit DNA strand representing the acceptable number of
%               standard deviations to keep processing.
%               00: Keep the entire image
%               01: Remove everything beyond 2 SD
%               10: Remove everything beyond 1.5 SD
%               11: Remove everything beyond 1 SD
%
% Output Arguments:
% rData:        The data used during processing of the spectra.

function [rData] = A04b_removeOutliers(iData, rData, DNA)
% Compute the Standard Deviation tolerance limits based on the input DNA
% argument.
switch DNA
    case {0}
        SDLimit=0;
    case {1}
        SDLimit=2;
    case {2}
        SDLimit=1.5;
    case {3}
        SDLimit=1;
end
disp(['    - Removing outliers beyond ' num2str(SDLimit) ' standard deviations.']);
if SDLimit ~=0
    tmpNumImage=rData.NumImages;
    rrData.NumPixels=0;
    rrData.NumImages=0;
    for cntImage=1:tmpNumImage
        if rData.CmbImg(cntImage).useImage==1
            % For the current image - what is the value of the Standard Deviation.
            MU=sum((rData.CmbImg(cntImage).data))/rData.CmbImg(cntImage).numPixels;
            tmpPixelNum=0;
            for cntPixel=1:rData.CmbImg(cntImage).numPixels
                tmpPixelNum=tmpPixelNum+1;
                residual(tmpPixelNum)=sum((rData.CmbImg(cntImage).data(cntPixel,:)-
MU(1,:)).^2);
            end % Pixel Count
            MD=mean(residual);
            SD=std(residual);
            minLimit = MD-(SDLimit*SD);
            maxLimit = MD+(SDLimit*SD);

            % Find number of pixels outside of the SD limits: Update the pixel
            % map and zero those data ranges not used.
            cntGoodPixels=0;
            tmpPixelNum=0;
            for cntPixel=1:rData.CmbImg(cntImage).numPixels
                tmpPixelNum=tmpPixelNum+1;
                if (residual(tmpPixelNum)<minLimit) || (residual(tmpPixelNum)>maxLimit)
                    rData.CmbImg(cntImage).data(cntPixel,1:iData.NumDataPoints)=0;
                else
                    cntGoodPixels=cntGoodPixels+1;
                    rrData.CmbImg(cntImage).data(cntGoodPixels,1:iData.NumDataPoints)=...
                        rData.CmbImg(cntImage).data(cntPixel,1:iData.NumDataPoints);
                end
            end % Pixel Count
            clear residual;
            clear MD;
            clear SD;
            rrData.CmbImg(cntImage).numPixels=cntGoodPixels;
            if rrData.CmbImg(cntImage).numPixels==0
                rrData.CmbImg(cntImage).useImage=0;
            else
                rrData.NumImages=rrData.NumImages+1;
                rrData.CmbImg(cntImage).useImage=1;
```

```
            end
            rrData.NumPixels=rrData.NumPixels+cntGoodPixels;
        end % Is image good
    end % Image Count
    rrData.NumDataPoints=rDData.NumDataPoints;
    rrData.Wavenumbers=rData.Wavenumbers;
    clear rData;
    rData=rrData;
else
    rData.NumImages=rData.NumImages-1+1;
end
```

6.7 Image Smoothing Function

```
% Tom Pinchuk
%
% Modified 10/02/2005
%
% This function is used to smooth the data in the images in reference to
% section 4.1.2 of the literature review.
%
% Images are smoothed using a 4-bit input parameter
%
% Assumptions:
%
%
% Input Arguments:
% iData:        The initial specatral data as determined by the load data
%               algorithm.
% rData:        The running Data up until now.
% DNA:          The 4-bit DNA strand representing two - 2-bit pieces of
%               information, the first being the method, and the second
%               being the number of points to use:
%               Bits 4-3:
%               00: No Smoothing
%               01: Boxcar
%               10: Mean
%               11: Savitzky-Golay
%               Bits 2-1:
%               00: 5-Points
%               01: 7-Points
%               10: 9-Points
%               11: 11-Points
%
% Output Arguments:
% rData:        The data used during processing of the spectra.

function [rData] = A06_SmoothImages(iData, rData, DNA)

% Split the DNA up into the two smoothing parameters.
smoothMethod = bitshift(DNA,-2);
tmpNum = DNA - bitshift(smoothMethod,2);
switch tmpNum
    case {0}
        smoothPoints=5;
    case (1)
        smoothPoints=7;
    case {2}
        smoothPoints=9;
    case {3}
        smoothPoints=11;
end
tmpPointsSide = (smoothPoints - 1) / 2;
switch smoothMethod
    case (1)
        disp(['    - Smoothing using Boxcar method and ' num2str(smoothPoints) '
points']);
    case {2}
        disp(['    - Smoothing using Mean method and ' num2str(smoothPoints) ' points']);
```

```
    case {3}
        disp(['    - Smoothing using Savitzky-Golay method and ' num2str(smoothPoints) '
points']);
end

%Boxcar smoothing - Remove end effects by getting rid of first and last
%                   points.
if smoothMethod == 1
    for tmpImageNum=1:iData.NumImages
        if rData.CmbImg(tmpImageNum).useImage==1
            boolNotEnd = 1;
            tmpOldIndex = tmpPointsSide + 1;
            tmpNewIndex = 0;
            while boolNotEnd==1
                tmpNewIndex = tmpNewIndex + 1;
                for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
                    newImg(tmpImageNum).data(cntPixels, tmpNewIndex) = ...
                        sum( rData.CmbImg(tmpImageNum).data(cntPixels, (tmpOldIndex-
tmpPointsSide):(tmpOldIndex+tmpPointsSide)) )...
                        / smoothPoints;
                end
                newImg(tmpImageNum).sampleDataPoints(tmpNewIndex) =
iData.Wavenumbers(tmpOldIndex);
                if (tmpOldIndex + smoothPoints + tmpPointsSide) >= iData.NumDataPoints
                    boolNotEnd = 0;
                else
                    tmpOldIndex = tmpOldIndex + smoothPoints;
                end
            end % While Loop
            newImg(tmpImageNum).NumDataPoints = tmpNewIndex;
        end % image Good
    end %counting the number of images
%Mean smoothing - Remove end effects by getting rid of first and last points.
elseif smoothMethod == 2
    for tmpImageNum=1:iData.NumImages
        if rData.CmbImg(tmpImageNum).useImage==1
            boolNotEnd = 1;
            tmpOldIndex = tmpPointsSide + 1;
            tmpNewIndex = 0;
            while boolNotEnd==1
                tmpNewIndex = tmpNewIndex + 1;
                for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
                    newImg(tmpImageNum).data(cntPixels, tmpNewIndex) = ...
                        sum( rData.CmbImg(tmpImageNum).data(cntPixels, (tmpOldIndex-
tmpPointsSide):(tmpOldIndex+tmpPointsSide)) )...
                        / smoothPoints;
                end
                newImg(tmpImageNum).sampleDataPoints(tmpNewIndex) =
iData.Wavenumbers(tmpOldIndex);
                if (tmpOldIndex + tmpPointsSide) >= iData.NumDataPoints
                    boolNotEnd = 0;
                else
                    tmpOldIndex = tmpOldIndex + 1;
                end
            end % while loop
            newImg(tmpImageNum).NumDataPoints = tmpNewIndex;
        end % image is good
    end %counting the number of images
%Savitzky - Golay smoothing - Remove end effects by getting rid of first and last points.
elseif (smoothMethod == 3)
    if smoothPoints==5, SGmult=[-3 12 17 12 -3]; end
    if smoothPoints==7, SGmult=[-2 3 6 7 6 3 2]; end
    if smoothPoints==9, SGmult=[-21 14 39 54 59 54 39 14 -21]; end
    if smoothPoints==11, SGmult=[-36 9 44 69 84 89 84 69 44 9 -36]; end
    sumSGmult = sum(SGmult);
    for tmpImageNum=1:iData.NumImages
        if rData.CmbImg(tmpImageNum).useImage==1
            boolNotEnd = 1;
            tmpOldIndex = tmpPointsSide + 1;
            tmpNewIndex = 0;
            while boolNotEnd==1
```

```
                tmpNewIndex = tmpNewIndex + 1;
                for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
                    tmpPoint(1:smoothPoints) = rData.CmbImg(tmpImageNum).data(cntPixels,
(tmpOldIndex-tmpPointsSide):(tmpOldIndex+tmpPointsSide));
                    tmpArg = tmpPoint.*SGmult;
                    newImg(tmpImageNum).data(cntPixels, tmpNewIndex) = sum(tmpArg) /
sumSGmult;
                end
                newImg(tmpImageNum).sampleDataPoints(tmpNewIndex) =
iData.Wavenumbers(tmpOldIndex);
                if (tmpOldIndex + tmpPointsSide) >= iData.NumDataPoints
                    boolNotEnd = 0;
                else
                    tmpOldIndex = tmpOldIndex + 1;
                end
            end % while loop
            newImg(tmpImageNum).NumDataPoints = tmpNewIndex;
        end %image is good
    end %counting the number of images
end

% If smoothing was done, update the rData file.
if smoothMethod>=1
    for cntImg=1:iData.NumImages
        if rData.CmbImg(cntImg).useImage==1
            rData.CmbImg(cntImg).data=newImg(cntImg).data;
            rData.NumDataPoints=newImg(cntImg).NumDataPoints;
            rData.Wavenumbers=newImg(cntImg).sampleDataPoints;
        end
    end
end
```

6.8 Feature Selection Function

```
% Tom Pinchuk
%
% Modified 10/08/2005
%
% This function is used to perform a feature selection proceadure on the
% sample data in accordance with section 5.2.x of the literature review.
%
% Assumptions:
%
%
% Input Arguments:
% iData:        The initial specatral data as determined by the load data
%               algorithm.
% rData:        The running Data up until now.
% DNA:          The 6-bit DNA strand representing the feature selection
%               algorithms to be applied.
%   ---------------------------------------------------------------------
%   |           |           |           |           |           |           |
%   |     6     |     5     |     4     |     3     |     2     |     1     |
%   |           |           |           |           |           |           |
%   ---------------------------------------------------------------------
%   |                                                                       |
%   |  Bit 6: Integration               Bit 3: Use Norm [1=On 0=Off]        |
%   |  Bit 5: Use Deriv [1=On 0=Off]    Bit 2: Norm Method [0=1 or 1=2]     |
%   |  Bit 4: Deriv Meth [0=Fst/1=Sed]  Bit 1: Mean Centering               |
%   |                                                                       |
%   ---------------------------------------------------------------------
%
% Output Arguments:
% rData:        The data used during processing of the spectra.
%

function [rData] = A07_FeatureSelection(iData, rData, DNA)

%Mean Centering - Calculate the mean of each data point in each sample
```

```
%                within an image and subtract that value from each of the
%                data points in the respective image.
if bitget(DNA,1) == 1
    disp(['    - Mean Centering the data']);
    for tmpImageNum=1:iData.NumImages
        if rData.CmbImg(tmpImageNum).useImage==1
            tempMean=mean(rData.CmbImg(tmpImageNum).data);
            for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
                rData.CmbImg(tmpImageNum).data(cntPixels,:) =...
                    rData.CmbImg(tmpImageNum).data(cntPixels,:) - tempMean;
            end
        end % Is image in use
    end
end

% Should we normalize Yes or No?
if bitget(DNA,3) == 1
    disp(['    - Normalization of data']);
    for tmpImageNum=1:iData.NumImages
        if rData.CmbImg(tmpImageNum).useImage==1
            if bitget(DNA,2) == 0
            % 1 - normalization - Normalize data in a particular spectrum to unit area.
            %                     Calculating the 1-Norm constant and dividing each and
            %                     every data point by that value
                for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
                    Norm1=sum(abs(rData.CmbImg(tmpImageNum).data(cntPixels, :)));
                    rData.CmbImg(tmpImageNum).data(cntPixels, :)=...
                        rData.CmbImg(tmpImageNum).data(cntPixels, :) ./ Norm1;
                end
            else
            % 2 - normalization - Normalize data in a particular spectrum to unit area.
            %                     Calculating the 2-Norm constant and dividing each and
            %                     every data point by that value
                for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
                    Norm2=sqrt(sum( (rData.CmbImg(tmpImageNum).data(cntPixels, :).^2) ));
                    rData.CmbImg(tmpImageNum).data(cntPixels, :)=...
                        rData.CmbImg(tmpImageNum).data(cntPixels, :) ./ Norm2;
                end %count pixels
            end % End if
        end % Is image in use
    end % Count Images
end %Normalize?


% Use Derivatives?
if bitget(DNA,5) == 1
    disp(['    - Derivation of data']);
    rrData.NumPixels=rData.NumPixels;
    rrData.NumImages=rData.NumImages;
    rrData.NumDataPoints=rData.NumDataPoints-2;
    rrData.Wavenumbers(1:rrData.NumDataPoints)=rData.Wavenumbers(2:(rData.NumDataPoints-
1));
    tmpResConstA = 1/(2*iData.Resolution);
    tmpResConstB = 1/(iData.Resolution^2);
    for tmpImageNum=1:iData.NumImages
        rrData.CmbImg(tmpImageNum).numPixels=rData.CmbImg(tmpImageNum).numPixels;
        rrData.CmbImg(tmpImageNum).useImage=rData.CmbImg(tmpImageNum).useImage;
        if rData.CmbImg(tmpImageNum).useImage==1
            if bitget(DNA,4) == 0
            % First Derivative -  First derivative is calculated and the total data
points
            %                     are adjusted accordingly.
                for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
                    %for cntDataPoints=1:rrData.NumDataPoints
                    %    rrData.CmbImg(tmpImageNum).data(cntPixels, cntDataPoints)=...
                    %        (rData.CmbImg(tmpImageNum).data(cntPixels, cntDataPoints)...
                    %        -rData.CmbImg(tmpImageNum).data(cntPixels,
(cntDataPoints+2)))...
                    %        *tmpResConstA;
                    %end %count data points
```

```
                    rrData.CmbImg(tmpImageNum).data(cntPixels,
1:rrData.NumDataPoints)=...
                                (rData.CmbImg(tmpImageNum).data(cntPixels,
1:rrData.NumDataPoints)...
                                -rData.CmbImg(tmpImageNum).data(cntPixels,
3:(rrData.NumDataPoints+2)))...
                                .*tmpResConstA;
                        end %count pixels
                    else
                    % Second Derivative - Second derivative is calculated and the total data
points
                    %                       are adjusted accordingly.
                        for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
                        for cntDataPoints=1:rrData.NumDataPoints
                            rrData.CmbImg(tmpImageNum).data(cntPixels, cntDataPoints)=...
                                (rData.CmbImg(tmpImageNum).data(cntPixels, cntDataPoints)...
                                -2*rData.CmbImg(tmpImageNum).data(cntPixels,
cntDataPoints+1)...
                                -rData.CmbImg(tmpImageNum).data(cntPixels,
(cntDataPoints+2)))...
                                *tmpResConstB;
                        end %count data points
                        end %count pixels
                    end % First / Second Method
                end %Is image good
            end %Count images
            rData=rrData;
            clear rrData;
        end % Do derivative?

        % Integration - Integrals are calculated based on the Simpson's method.
        %               The total data points are adjusted accordingly.
        if bitget(DNA,6) == 1
            disp(['    - Integration of data']);
            rrData.NumPixels=rData.NumPixels;
            rrData.NumImages=rData.NumImages;
            rrData.NumDataPoints=rData.NumDataPoints-1;
            rrData.Wavenumbers(1:rrData.NumDataPoints)=rData.Wavenumbers(1:(rData.NumDataPoints-
1))+(iData.Resolution/2);
            tmpResConst=iData.Resolution/6;
            for tmpImageNum=1:iData.NumImages
                rrData.CmbImg(tmpImageNum).numPixels=rData.CmbImg(tmpImageNum).numPixels;
                rrData.CmbImg(tmpImageNum).useImage=rData.CmbImg(tmpImageNum).useImage;
                if rData.CmbImg(tmpImageNum).useImage==1
                    for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
                        for cntDataPoints=1:rrData.NumDataPoints
                            rrData.CmbImg(tmpImageNum).data(cntPixels, cntDataPoints)=...
                                (1.5*rData.CmbImg(tmpImageNum).data(cntPixels, cntDataPoints)+...
                                1.5*rData.CmbImg(tmpImageNum).data(cntPixels,
cntDataPoints+1))*tmpResConst;
                        end % Count data points
                    end %count pixels
                end %Is image good
            end % count image number
            rData=rrData;
            clear rrData;
        end %end if
```

6.9 Image Combination Function

```
% Tom Pinchuk
%
% Modified 10/08/2005
%
% This function is used to combine the image data using the greatest
% resolution into one single file.
%
% Assumptions:
%
```

```
%
% Input Arguments:
% iData:        The initial specatral data as determined by the load data
%               algorithm.
% rData:        The running Data up until now.
%
% Output Arguments:
% rData:        The data used during processing of the spectra.
%
function [rData] = A08_CombineImageData(iData, rData)
disp(['    - Combining the data']);
cntSpec=0;
rrData.NumPixels=rData.NumPixels;
rrData.NumImages=rData.NumImages;
rrData.NumDataPoints=rData.NumDataPoints;
rrData.Wavenumbers=rData.Wavenumbers;
for tmpImageNum=1:iData.NumImages
    if rData.CmbImg(tmpImageNum).useImage==1
        for cntPixels=1:rData.CmbImg(tmpImageNum).numPixels
            cntSpec=cntSpec+1;
            rrData.TotImg.class(cntSpec)=iData.Image(tmpImageNum).class;
            rrData.TotImg.data(cntSpec,:)=rData.CmbImg(tmpImageNum).data(cntPixels,:);
            rrData.TotImg.imgnum(cntSpec)=tmpImageNum;
        end %count pixels
    end %Use Image
end %number of files
rrData.NumClasses=length(unique(rrData.TotImg.class));
rData=rrData;
```

6.10 Principal Component Analysis Function

```
% Tom Pinchuk
%
% Modified 10/08/2005
%
% This function is used to determine the principal components of the data
% set for utilization in classification proceedures.
%
% Assumptions:
%
%
% Input Arguments:
% iData:        The initial specatral data as determined by the load data
%               algorithm.
% rData:        The running Data up until now.
% DNA:          The 4-bit DNA strand representing the number of Principal
%               Components from 0 (use all data) to 15 to use.
%
% Output Arguments:
% rData:        The data used during processing of the spectra.
%

function [rData, pc] = A09_PCA(iData, rData, DNA)
if DNA > 1
    disp(['    - Calculating Principal Coponents: ' num2str(DNA)]);
    rrData.NumPixels=rData.NumPixels;
    rrData.NumImages=rData.NumImages;
    rrData.NumDataPoints=rData.NumDataPoints;
    rrData.Wavenumbers=rData.Wavenumbers;
    rrData.NumClasses=rData.NumClasses;
    rrData.TotImg.class=rData.TotImg.class;
    rrData.TotImg.imgnum=rData.TotImg.imgnum;

    covMatrix = cov(rData.TotImg.data);
    [pc,variances,explained] = pcacov(covMatrix);
    for cntPC = 1:DNA
        tempMult = pc(cntPC,:)';
        for cntPixel = 1:rData.NumPixels
```

```
            rrData.TotImg.data(cntPixel,cntPC) = rData.TotImg.data(cntPixel,:) *
tempMult;
        end
    end % counting data points

    rData=rrData;
else
    pc=0;
end
```

6.11 Cluster Function

```
% Tom Pinchuk
%
% Modified 10/08/2005
%
% This function is the newest clustering function. It works via a linkage
% to determine the distance between clusters.
%
% Assumptions:
% B version is for kMeans
%
% Input Arguments:
% rData:        The running Data up until now.
%
% Output Arguments:
% rData:        The data used during processing of the spectra.

function [rData] = A10b_ClusterFunction(rData)
txtDistMethod = 'sqEuclidean';
txtStartMethod = 'sample';
cstNumIterations = 5;
%disp(['    - CLUSTERING: Distance Function ' ]);
%distFunc=pdist(rData.TotImg.data);
rrData.class=rData.TotImg.class;
rrData.NumClasses=rData.NumClasses;
rrData.NumPixels=rData.NumPixels;
%clear rData;
%disp(['    - CLUSTERING: Linkage' ]);
%link=linkage(distFunc);
%clear distFunc;
disp(['    - CLUSTERING: Calculating Clusters' ]);
[clusterOut.category, clusterOut.centroid, clusterOut.WCSP, clusterOut.distToClusters]
=...
    kmeans(rData.TotImg.data,
rrData.NumClasses,'distance',txtDistMethod,'start',txtStartMethod,...
    'replicates',cstNumIterations,'emptyaction','singleton');
%clust = cluster(link,'maxclust',rrData.NumClasses);
%sqDist=squareform(distFunc);
%[H,T,perm] = dendrogram(link,0,'colorthreshold','default');
clear rData;

cntClass(1:rrData.NumClasses)=0;
% Rows are original class, columns are number of instances of new class.
disp(['    - CLUSTERING: Validating and Scoring' ]);
newClass(1:rrData.NumClasses,1:rrData.NumClasses)=0;
for cntPixels=1:rrData.NumPixels
    cntClass(rrData.class(cntPixels))=cntClass(rrData.class(cntPixels))+1;
    newClass(rrData.class(cntPixels),clusterOut.category(cntPixels))=...
        newClass(rrData.class(cntPixels),clusterOut.category(cntPixels))+1;
end
tmpClass=newClass;
for cntCC=1:rrData.NumClasses
    % Get the maximum for each column and the row index's where they occur
    [colMax, rowIndexs]=max(tmpClass);
    % Get the maximum from the column results and the column index where it occurs
    [absMax, colIndex]=max(colMax);
    % Get the row index
    rowIndex=rowIndexs(colIndex);
```

```
    % MAp the class
    mapClass(rowIndex)=colIndex;
    % Now that it is found, make sure that neither of those are used again,
    % zero out the row and columns.
    tmpClass(rowIndex,1:rrData.NumClasses)=-100;
    tmpClass(1:rrData.NumClasses,colIndex)=-100;
end

%Determine if the classes are correct.
cntCorrect=0;
for cntPixels=1:rrData.NumPixels
    if clusterOut.category(cntPixels)==mapClass(rrData.class(cntPixels))
        cntCorrect=cntCorrect+1;
    end
end
rData.NumPixels=rrData.NumPixels;
rData.Score = cntCorrect/rrData.NumPixels;
```

6.12 Reproduction Function

```
%
% Tom Pinchuk
%
% Function Reproduce: Modified 9/30/2005
% This function creates 2 new strands of DNA from the existing population
% taking into account several factors. It will randomly assign DNA to make
% the initial population. It will also verify the strand to ensure
% continuity and no duplicates.
%
% Switch the bOdd value if their is only one DNA strand found.

function [DNA, bOdd, oData] = reproduce(cntIter, oData, bOdd)
format long e
% This section contains constants
cstInitPop = 50;     % Initial population to randomize and select from
cstSelectExp = 2;    % The exponential factor for selecting "Parents" randomly.
cstPctRandom = 7;    % The random percentage factor for creating a "random"
                     % parent for reproduction
cstPctMutate = 7;    % The random percentage factor for mutating a "random"
                     % child after reproduction

% Rank order the results till now. Fill in the oData.Rank(iteration) field
% with the actual rank of the scores.
[tmpScores, tmpRank]=sort(oData.totScore);
for cntRank=1:cntIter-1
    tmpIter = tmpRank(cntIter-cntRank);
    oData.Rank(tmpIter)=cntRank;
end

% Create initial Population
if cntIter <= cstInitPop
    isGoodDNA=0;
    while isGoodDNA==0
        isGoodDNA = 1;
        isDNA1Good = 1;
        isDNA2Good = 1;
        DNA(1)=round(rand*134217728);
        DNA(2)=round(rand*134217728);
        if not(isempty(find(oData.DNA==DNA(1))))
            DNA(1)=-1;
            isDNA1Good=0;
        end
        if not(isempty(find(oData.DNA==DNA(2))))
            DNA(2)=-1;
            isDNA2Good=0;
        end
        if DNA(2) == DNA(1), isDNA2Good=0; end
        if (isDNA1Good==0) && (isDNA2Good==1)
            DNA(1)=DNA(2);
```

```
            if bOdd==0, bOdd=1; else bOdd=0; end
        elseif (isDNA1Good==1) && (isDNA2Good==0)
            if bOdd==0, bOdd=1; else bOdd=0; end
        elseif (isDNA1Good==0) && (isDNA2Good==0)
            isGoodDNA=0;
        end
    end
else
    % Here is the real GENETIC ALGORITHM once the population has been
    % verified. If a child has been found, make child value = -1. The
    % maximum population to reproduce is the same as the initial
    % population.
    isGoodDNA=0;
    while isGoodDNA==0
        isGoodDNA = 1;
        isDNA1Good = 1;
        isDNA2Good = 1;
        % 1: Determine the top performers in the population and randomly select
        %    2 of them as parents. The top performers are first sorted by
        %    the percentage clasified and then by the total score. The
        %    selection is based on an exponential curve fit. There is a
        %    lower probablility of selecting lower scored samples and a
        %    higher of slecting higher scores samples. Y = X^N
        %    ROUND((RAND)^(1/N) x Max Population) = Index
        PICK(1:2)=0;
        % Pick two parents from top population 0 to 50
        while PICK(1)==PICK(2)
            PICK(1:2)=round((rand(1:2).^(cstSelectExp)).*(cstInitPop-1));
        end
        rnkIndex(1)=tmpRank(length(oData.Rank)-PICK(1));
        rnkIndex(2)=tmpRank(length(oData.Rank)-PICK(2));
        PARENT(1) = oData.DNA(rnkIndex(1));
        txtParentPct(1) = tmpScores((length(oData.Rank)-PICK(1)));
        %txtParentPxl(1) = tmpScores((length(oData.Rank)-PICK(1)),2);
        % Create a random parent if the random process falls into a given
        % percentage.
        if (rand*100) <= cstPctRandom
            PARENT(2) = round(rand*134217728);
            txtParentPct(2) = -1;
        else
            PARENT(2) = oData.DNA(rnkIndex(2));
            txtParentPct(2) = tmpScores((length(oData.Rank)-PICK(2)));
            %txtParentPxl(2) = tmpScores((length(oData.Rank)-PICK(2)),2);
        end

        % 2: Randomly pick a point where the cut to the right will provide for
        %    crossover reproduction. The bit must be to the right of bit 27
        %    through bit 2 in order to have children.
        tmpShiftBit = (round(rand*25) + 2);
        tmpFirstCut=bitshift(PARENT,-tmpShiftBit);
        tmpSecondCut=PARENT - bitshift(tmpFirstCut,tmpShiftBit);
        DNA(1) = bitshift(tmpFirstCut(1),tmpShiftBit) + tmpSecondCut(2);
        DNA(2) = bitshift(tmpFirstCut(2),tmpShiftBit) + tmpSecondCut(1);

        % 3: Randomly select if mutation should be applied. If so, randomly
        %    select a sample and randomly select a bit to mutate.
        tmpMutateBit = 0;
        if (rand*100) <= cstPctMutate
            tmpMutateBit = round(rand*26) + 1;
            tmpMutateChild = round(rand) + 1;
            if bitget(DNA(tmpMutateChild),tmpMutateBit)==1
                DNA(tmpMutateChild)=bitset(DNA(tmpMutateChild),tmpMutateBit,0);
            else
                DNA(tmpMutateChild)=bitset(DNA(tmpMutateChild),tmpMutateBit,1);
            end
        end %If mutateable

        % Verify the children
        if not(isempty(find(oData.DNA==DNA(1))))
            DNA(1)=-1;
            isDNA1Good=0;
```

```
        end
        if not(isempty(find(oData.DNA==DNA(2))))
            DNA(2)=-1;
            isDNA2Good=0;
        end
        if DNA(2) == DNA(1), isDNA2Good=0; end
        if (isDNA1Good==0) && (isDNA2Good==1)
            DNA(1)=DNA(2);
            if bOdd==0, bOdd=1; else bOdd=0; end
        elseif (isDNA1Good==1) && (isDNA2Good==0)
            if bOdd==0, bOdd=1; else bOdd=0; end
        elseif (isDNA1Good==0) && (isDNA2Good==0)
            isGoodDNA=0;
        end
    end % While Loop

    % Display what is going on:

    disp(['   Cutting parents to the right of bit: ' num2str(tmpShiftBit)]);
    if tmpMutateBit > 0
    disp(['      Mutating Bit: ' num2str(tmpMutateBit) ' in child number'
num2str(tmpMutateChild)]);
    end
        disp(['      Parent 1: ' num2str(PARENT(1)) ' Score: '
num2str(txtParentPct(1)*100) '% ']);
    if txtParentPct(2) == -1
        disp(['      Parent 2: ' num2str(PARENT(2)) ' - RANDOMLY GENERATED']);
    else
        disp(['      Parent 2: ' num2str(PARENT(2)) ' Score: '
num2str(txtParentPct(2)*100) '% ']);
    end
    disp(char(160));
    firstHalf(1,:)=27:-1:tmpShiftBit;
    firstHalf(2,:)=bitget(PARENT(1),27:-1:tmpShiftBit);
    firstHalf(3,:)=bitget(PARENT(2),27:-1:tmpShiftBit);
    if DNA(1)==-1
        firstHalf(4,:)=27:-1:tmpShiftBit;
        secondHalf(4,:)=(tmpShiftBit-1):-1:1;
    else
        firstHalf(4,:)=bitget(DNA(1),27:-1:tmpShiftBit);
        secondHalf(4,:)=bitget(DNA(1),(tmpShiftBit-1):-1:1);
    end
    if DNA(2)==-1
        firstHalf(5,:)=27:-1:tmpShiftBit;
        secondHalf(5,:)=(tmpShiftBit-1):-1:1;
    else
        firstHalf(5,:)=bitget(DNA(2),27:-1:tmpShiftBit);
        secondHalf(5,:)=bitget(DNA(2),(tmpShiftBit-1):-1:1);
    end
    secondHalf(1,:)=(tmpShiftBit-1):-1:1;
    secondHalf(2,:)=bitget(PARENT(1),(tmpShiftBit-1):-1:1);
    secondHalf(3,:)=bitget(PARENT(2),(tmpShiftBit-1):-1:1);
    middlePart=['<=>';'<=>';'<=>';'<=>';'<=>'];
    prePart=['        Bit:';'   Parent 1:';'   Parent 2:';'    Child 1:';'    Child 2:'];
    disp([prePart(1,:) num2str(firstHalf(1,:)) ' ' middlePart(1,:) ' '
num2str(secondHalf(1,:))]);
    disp([prePart(2,:) num2str(firstHalf(2,:)) ' ' middlePart(2,:) ' '
num2str(secondHalf(2,:))]);
    disp([prePart(3,:) num2str(firstHalf(3,:)) ' ' middlePart(3,:) ' '
num2str(secondHalf(3,:))]);
    disp([prePart(4,:) num2str(firstHalf(4,:)) ' ' middlePart(4,:) ' '
num2str(secondHalf(4,:))]);
    disp([prePart(5,:) num2str(firstHalf(5,:)) ' ' middlePart(5,:) ' '
num2str(secondHalf(5,:))]);
end
```