Multivariate Analysis of FT-IR Microspectroscopic Data of Abdominal Aortic Aneurysm in Mice using Principal Component Analysis and Gaussian Mixture Model

Honors Research Thesis

Presented in Partial Fulfillment of the Requirements for Graduation “with Honors Research Distinction in Chemistry” in the Undergraduate Colleges of The Ohio State University

By

Olla Nayal

The Ohio State University

May 2013

Project Advisor: J. Clay Harris, Department of Chemistry and Biochemistry
Table of Contents

Abstract 2
Introduction 3
Experimental 10
  A. Tissue Samples 10
  B. Instrumentation 11
  C. Analytical Software 12
  D. Data Analysis 14
    i. Background Removal 14
    ii. Tissue Analysis 14
Linear Discriminant Analysis for Image Segmentation Results and Discussion 15
Linear Discriminant Analysis for Image Segmentation Conclusions 15
Gaussian Mixture Model for Image Segmentation Results and Discussion 20
Gaussian Mixture Model for Image Segmentation Conclusions 27
Gaussian Mixture Model for Tissue Analysis Results and Discussion 27
Gaussian Mixture Model for Tissue Analysis Conclusions 34
References 35
Appendix 38
Abstract

Fourier transform infrared (FT-IR) microspectroscopy enables high spatial resolution biochemical analysis and imaging of tissue. This research utilizes principal component analysis (PCA) and Gaussian mixture models (GMM) to analyze murine abdominal aorta samples of both healthy and diseased tissue with an abdominal aortic aneurysm (AAA) present. The samples were mounted onto glass slides and analyzed via reflectance FT-IR microspectroscopy. The spectra collected per sample numbered in the thousands, with hundreds of data points contained within a spectrum. The size of the hyperspectral data sets and their continuous nature require analysis by robust multivariate techniques like PCA. PCA is widely used to reduce multivariate data into a few dimensions (PCs) that incorporate most of the variance in the data. For more effective analysis of the tissue, GMM was used following PCA to separate tissue spectra from spectra of the glass slide. GMM is an unsupervised clustering technique that assumes a finite number of Gaussian distributions within a data set. Prior application of PCA was found to be critical to successful GMM clustering of the data for removal of background data. GMM clustering was subsequently applied to isolated tissue spectra preprocessed with PCA for various infrared regions to determine which bands were useful for distinguishing healthy and diseased samples. PC 2 provided more accurate clustering of tissue spectra into 2 classes of diseased and healthy. The technique advanced by this research is useful for determining infrared bands useful for distinguishing healthy arterial tissue and tissue affected by AAA.
Introduction

“An aneurysm is a permanent focal dilation of an artery to 1.5 times its normal diameter.” Aneurysm of the abdominal aorta, the largest artery in the abdominal cavity, is termed abdominal aortic aneurysm (AAA). This condition mostly affects the elderly and is responsible for 15,000 deaths a year in the United States alone. In the year 2000, AAA was the 10th leading cause of death in American white males age 65 to 74 years. Most patients with AAA present no symptoms and the condition is often detected on studies performed for other purposes using medical imaging techniques like ultrasound and computed tomography (CT). Ultrasound screening has a sensitivity and specificity of 100 and 96 percent, respectively, for detecting certain types of AAA and is recommended for high-risk individuals. Risk factors include age over 65 years, male sex, smoking at least 100 cigarettes in a lifetime, and family history. Studies have shown that 12-19% of first-degree relatives of a patient with AAA develop an aneurysm. When symptoms are present, they include back, abdominal, or groin pain and the presence of a pulsating abdominal mass. Symptomatic patients are evaluated immediately using CT to determine the size of the aneurysm. While ultrasound and CT are effective procedures for diagnosing aneurysm upon onset, no reliable and practical methods exist for evaluating the risk of aneurysm well in advance of incidence.

Figure 1. Cartoon representation of an abdominal aortic aneurysm (AAA)
Several events have been linked to the development of AAA, the main one being the proteolytic degradation of the extracellular matrix proteins elastin and collagen. Elastin and collagen are the main structural components of blood vessel walls, and changes in the distribution of these proteins is present in aneurysms as well as atherosclerosis. A decrease in the elastin concentration in aortic walls along with an increase in collagen concentration have been widely reported in humans with AAA. Studies have shown increased local production of metalloproteinases (MMPs)—enzymes capable of degrading elastin and collagen—in patients with AAA. Significant correlation has been shown between aneurysm size and expression of certain MMPs. In addition, it has been proposed that AAA is an antigen-driven autoimmune disease. Extensive lymphocytic and monocytic infiltration of the aortic wall with deposition of immunoglobulin G has been connected to AAA development. Furthermore, cigarette smoking has been linked to an increased inflammatory response within the aorta wall by affecting immune-mediated pathways. The formation and rupture of aneurysms is also attributed to increased biomechanical wall stress, which explains why the thoracic aorta is relatively resistant to aneurysm enlargement compared to the abdominal aorta considering hemodynamic flow field and biomechanical forces applied to these different regions.

Studies of AAA pathogenesis indicate clear biochemical markers associated with the development of aneurysm, specifically changes in collagen and elastin distribution in arterial walls. The biochemical differences between normal abdominal aortas and aneurismal aortas are exploited in this investigation to discern healthy and diseased tissue using Fourier transform infrared spectroscopy (FT-IR). FT-IR measures the interaction of molecular bonds with different wavelengths of infrared light and provides a “biochemical fingerprint” of tissue samples. FT-IR has been used to detect collagen deposition in cardiac tissue after myocardial infarction, commonly known as a heart attack. While immunohistochemical assays are sensitive
quantitative methods for visualizing excess collagen deposition in cardiac tissue, they are costly and labor intensive. Deposition of fibrillar collagen is found in subjects with heart attacks, and similar extracellular matrix restructuring is present in other cardiac pathologies, including AAA. The infrared absorbance band at 1338 cm\(^{-1}\) based on the C-H wagging vibration of proline side chains determined by earlier studies was used to map collagen deposition in rat heart tissue. This wavenumber was also characteristic of other collagen types including type II, type III, and type IV. While other infrared absorbance bands such as the amide I carbonyl stretching vibration were examined, only the band at 1338 cm\(^{-1}\) was found to correspond to collagen content. There was a strong linear correlation between immunohistochemical staining of type I collagen and intensity of the band at 1338 cm\(^{-1}\).

The concentration of radiation absorbing species in a solution is related to absorbance intensity by the Beer-Lambert law, also known as Beer's law. Beer's law is used for quantifying molecular concentrations measured by infrared spectroscopy and other spectroscopic techniques. Absorbance is directly related to molecular concentration and radiation path length. The thickness of a given sample provides a path for light to travel and may result in scattering of radiation, which would impact absorbance intensity. Investigators in the previously mentioned study assumed that because they were working with absorbance units less than or equal to 1, they could neglect the effects of sample thickness. Peak integration about 1338 cm\(^{-1}\) was used to generate false color images mapping the collagen deposition in the rat tissue samples. In this investigation, a large fraction of the absorbance values contained within hyperspectral datasets of AAA samples are above 1, and thickness contributes significantly to these values.

Other studies have similarly relied on calibration models to quantify the structurally protein content of cardiac tissue. Urbas et al. employed diffuse reflection near-infrared (near-
IR) spectroscopy to determine the collagen-to-elastin ratio of mouse arterial tissue affected by AAA. A calibration model was developed based on correlation of near-IR spectra to collagen-to-elastin ratios of the tissue measured by scanning electron microscope (SEM) morphometry, a type of histological marker. Principal component analysis (PCA) and principal component regression (PCR) were used to reduce the hyperspectral data for construction of calibration models. PCA is an orthogonal linear transformation used to reduce multivariate data to a few dimensions (PCs) that contain most of the variance in the original data.

Mathematically, PCA is the eigenvalue decomposition of the covariance matrix of a multivariate dataset. The covariance between two variables in a dataset indicates how they vary relative to each other. If the covariance is positive, then the two variables are directly related and if it is negative they are inversely related. If the variables are independent, then the covariance will equal zero. The covariance matrix contains covariance values for all combinations of variable pairs of a multivariate dataset. Eigenvalue decomposition of the covariance matrix produces eigenvectors and eigenvalues. The eigenvectors are orthogonal and indicate patterns in the data. Each eigenvector has a corresponding eigenvalue, which quantifies the eigenvector’s significance. For example, the eigenvector with the highest eigenvalue is along the first PC, which accounts for the greatest variance in the data. By eigenvalue decomposition hyperdimensional data can be compressed into a few dimensions so that comparison of data points is easier. PCA is used in this investigation to reduce each individual infrared spectrum into a single data point. For a given PC, absorbance values at different wavenumbers are assigned different weights, which are the eigenvectors themselves. A linear combination of all absorbance values multiplied by their corresponding weights is calculated for each spectrum. This is taken as the new value of the data point in a particular PC. Each PC has its own set of weights, which are also called loadings, so a transformed data point
can have as many coordinates as PCs. The values of the transformed data are called scores. The maximum number of PCs that can be computed is equal to the number of wavenumbers in the original spectra. It is also important to subtract the mean from each data dimension so that PCA works properly.\textsuperscript{27}

**Figure 2.** Schematic of Principal Component Analysis. PC 1 lies along the greatest variance in the data. PC 2 lies along the second greatest variance in the data orthogonal to PC 1.\textsuperscript{28}

Urbas et al. relied on PCA and PCR to develop a calibration model from which collagen and elastin content in unknown tissue samples could be determined to indicate the health of the tissue. To construct the calibration model Urbas et al. used lyophilized collagen and elastin to prepare the standards that they subsequently used to study spectra of mouse abdominal aorta tissue. However, considering that tissue contains many organic components, species besides collagen and elastin would also contribute to the infrared spectrum of a tissue sample. Many organic species absorb in the same infrared regions, so it cannot be assumed that an absorbance in a particular region is due to only collagen and elastin. Furthermore, while changes in collagen and elastin content are considered the main biochemical changes associated with AAA, previously mentioned histochemical studies have implicated many other
phenomena in the development of AAA. These would likely also have an affect on tissue transformation and could be observed with infrared spectroscopy. The assertion that there is more taking place in the aortas than collagen/elastin changes is in fact recognized by Urbas et al.\textsuperscript{26} Rather than focus on a single chemical entity to measure, this investigation utilizes data clustering techniques to determine clustering success of samples from a control and experimental group. The control group consists of healthy mice while the experimental group consists of genetic knockout mice with AAA. AAA was induced in the experimental group by intravenous administration of angiotensin II (AngII).\textsuperscript{29}

While the spectra of the samples used in this investigation were collected from 720-4000 cm\textsuperscript{-1}, smaller regions were excised and analyzed with PCA to produce better clustering results and exclude regions like the O-H stretch due to water (3200-3500 cm\textsuperscript{-1}) that could group the data based on water content. Including this region may result in a large weight assigned to the wavenumbers in the O-H stretch region if water is a large source of variation in the samples. In addition, variations in thickness within the samples also contribute to absorbance values and may result in clustering based on sample thickness; recall that Beer's law relates absorbance to sample thickness.\textsuperscript{30} Examining the coefficients of the linear combination can show which wavenumbers were important for distinguishing between clusters. These wavenumbers can be correlated to known vibrational modes of functional groups belonging to different organic species.

Data clustering methods have been used in other studies to group healthy and diseased tissue based on their IR spectra. Martin et al. utilized PCA followed by linear discriminant analysis (LDA) to compare prostate tissue in an area with prostate adenocarcinoma (CaP) and prostate tissue without CaP.\textsuperscript{31} Linear discriminant analysis (LDA) is a supervised clustering technique that computes a linear combination from a training set of known data. This linear
combination can be used to cluster unknown data into a number of user-designated classes. LDA works best when only 2 classes are designated. The coefficients of the linear combination indicate the variable weights, thus revealing which wavenumbers contributed the most to successful clustering. Martin et al. applied PCA before LDA so that different PCs served as variables instead of individual wavenumbers.

In this investigation, clustering techniques were exploited not only for the purpose of discriminating between healthy and diseased tissue, but also for selecting tissue data to analyze apart from data of the slide on which the tissue samples were mounted. Prior to 2D FT-IR analysis, all tissue slices were frozen in Optimal Cutting Temperature compound (OCT) and sectioned and mounted onto glass slides. The output datasets included spectra of both the tissue and the slides. Since PCA is a data-driven technique, excluding background data from the slide should result in different scores for the tissue data. Thus, removal of slide data points was a critical step in this study.

Considering that the number of spectra for a single tissue slice mounted on a slide numbered in the thousands, manually removing data points would have been a long and arduous process. Thresholding, also referred to as masking, is a widely used imaging technique for selecting parts of an image. A cutoff is set and any values above or below that cutoff are excluded so that only the desired portion is left. In a NIR multivariate study to predict quality attributes of lamb meat, Kamruzzaman et al. used thresholding to isolate lamb meat from surrounding background and fat. Another threshold was applied to isolate only the fat. While thresholding may have been an effective approach for separating tissue and background data in this study, it would have required much user input and experimentation to establish an appropriate cutoff. Clustering techniques were a more attractive option because they could separate data quickly and automatically. In addition, it was seen as a good opportunity to test
the effectiveness of clustering techniques before they would be used to analyze tissue data. The assumption was that any methods that could not separate tissue and background data could not be expected to separate the more similar healthy and diseased tissue.

LDA was tested for this purpose, but selecting representative data points to input for a training set was a laborious and unreliable process. Therefore, unsupervised clustering algorithms like Gaussian mixture model (GMM) were investigated. GMM is one of many techniques used in image segmentation to delineate the boundaries of objects in an image. This process is often carried out during analysis of surveillance videos. An evaluation of several segmentation algorithms performed by Toyama et al. showed that GMM was effective for differentiating background and foreground pixels even in the presence of common problems associated with background subtraction. GMM operates under the assumption that a given dataset is composed of a number of user-specified Gaussian distributions. GMM is very similar to \( k \)-means, and \( k \)-means is a sub function in the GMM algorithm used in this investigation. Both make an initial guess of the centers for the number of expected Gaussian distribution input by the user. The initial guesses for the cluster centers are random. These centers are then iteratively refined as the data points shift between clusters until the algorithm converges.

**Experimental**

**Tissue Samples**

Tissue samples were obtained from both a healthy mouse and an apolipoprotein E (apoE) knockout mouse 2-12 months in age. An AAA was induced in the knockout mouse through infusion of angiotensin II. Refer to *New Bioinformatic Techniques for the Analysis of Large Datasets* for more information about sample preparation. Visible light microscope images for control and experimental samples are shown in Figure 3.
Figure 3. A visible light image of the first section of c-ak-1 (A), c-bk-1 (B), x-bk-1 (C), and x-ak-1 (D). The naming convention was c (control) or x (experimental) – ak (above kidney) or bk (below kidney) – section number.\textsuperscript{7}

**Instrumentation**

2D FT-IR reflectance spectra were obtained from 720-4000 cm\textsuperscript{-1} at 8 cm\textsuperscript{-1} spectral resolution using a Perkin-Elmer Spotlight model 300 with a globar source and focal plain array. Rectangular maps were collected for each tissue section with a pixel size of 6.25 x 6.25 µm. The format of the output data sets is represented by the schematic in Figure 4. Refer to *New Bioinformatic Techniques for the Analysis of Large Datasets* for more information about instrumentation.\textsuperscript{7}
Figure 4. Schematic of the data sets being analyzed. Each tissue sample lies in the xy plane shown above. Each 6.25 µm x 6.25 µm pixel contains an FT-IR spectrum in the 720-4000 cm$^{-1}$ range. Thus, each tissue sample is represented by a 3-dimensional matrix.\textsuperscript{36}

**Analytical Software**

Principal component analysis (PCA) was performed in Matlab 7.13.0.564 (The Math Works, Inc., Mass.) using the built in “princomp” function (See the appendix for built in MatLab functions, open source code, and written in house functions). This function accepts an $n$-by-$p$ data matrix $x$ and outputs an $n$-by-$p$ matrix of scores with $n$ observation and $p$ principal components (PCs). Note that an $a$-by-$b$ matrix has $a$ rows and $b$ columns. A $p$-by-$p$ matrix of coefficients is also output with the coefficients for a given PC contained in a single column. The maximum number of PCs possible is equal to the number of observational variables. The number of observations $n$ should be greater than the number of observational variables $p$ for PCA to be effective. The “princomp” function centers $x$ by subtracting off the column means. Eigenvalue decomposition is subsequently performed using the built in singular value decomposition function “svd”.

Linear discriminant analysis (LDA) was performed using open source code function “LDA.” “LDA” calculates linear discriminant coefficients from an $n$-by-$p$ input training set with $n$ observations and $p$ variables. An additional $n$-by-1 matrix of class labels denoting the class assignment of observations within the training set must be input. Zeros and ones are commonly used labels for a two-class system. The function outputs a $p$-by-$(p+1)$ matrix. The rows correspond to different classes and the columns to the coefficients of the linear combination for each class. The first column is the constants. For each $p$-coordinate point in a test dataset, the linear combination computed for each class yields the probability of belonging to that class. The point is assigned to the class with the higher probability, and this process is carried out for all points in the test dataset.

A graphical user interface (GUI) called “classificationLDA” was created to automate the process of training set input and test set classification. The training set and class labels must be separately constructed before input into the GUI. The classification function performs PCA on the data first and allows the user to select one or several PCs as variables. The output matrix indicates data points of the same cluster by the same integer.

The Gaussian mixture model (GMM) algorithm was performed using open source code function “GMM.” The number of iterations and anticipated clusters are input by the user. This function clusters a $p$-by-$n$ data matrix of $n$ observations and $p$ variables into the specified number of clusters. It indicates whether the algorithm converged after the specified number of iterations. $k$-means is a sub function of this algorithm. The output 1-by-$n$ matrix indicates data points of the same cluster by the same integer.
Data Analysis

Background Removal

The first analysis tested the effectiveness of LDA for discerning between tissue and slide pixels using “classificationLDA”. C-ak-1 was used as a training set. The training set classes were established by manually setting a threshold of 25000 transmittance units for the c-ak-1 matrix summed along the spectral dimension. Total transmittance values above the cutoff were assigned to the slide and those below the cutoff were assigned to the tissue. The following were input as test sets: c-ak-1, c-ak-2, c-ak-3, c-ak-4, c-ak-5. LDA classification was performed for PCs 1-3. The output class labels were reformatted into the original 2D layout of the tissue sections to indicate the class assignment of each pixel.

GMM clustering was performed in the second analysis both with and without pre-processing of the data sets with PCA. Tissue sections were individually analyzed. For data sets not pre-processed with PCA the observational variables were the absorbances at 720-4000 cm\(^{-1}\). For data sets pre-processed with PCA the observational variables were scores for selected PCs. PCs 1-3 were selected individually and in combinations for input into the GMM function. The algorithm was carried out with both 2 and 3 clusters specified and 10 iterations. The output cluster labels were reformatted into the original 2D layout of the tissue sections to indicate the cluster assignment of each pixel.

Tissue Analysis

For data sets clustered by GMM, the clusters belonging to the tissue were visually identified by comparison to the corresponding visible light microscope images. GMM was subsequently applied to an array of all the excised tissue data from the control and experimental group with the hope that the algorithm would be able to distinguish between the two groups. Only data from sections above the kidney branch were used because AAA only
developed in that area in the experimental group. IR regions from 720 to 4000 cm\(^{-1}\), 720 to 3104 cm\(^{-1}\), and 720 to 1504 cm\(^{-1}\) were each analyzed. In addition to analyzing the entire spectrum, the region from 720 to 3104 cm\(^{-1}\) was selected to exclude the peak due to water (3200-3500 cm\(^{-1}\)). The region from 720 to 1504 cm\(^{-1}\) was selected as the fingerprint region. Tissue data from c-ak-1 to c-ak-12 and x-ak-3 to x-ak-12 were concatenated into a single array and then analyzed with PCA. The total number of spectra was 37,988. Rows 1 through 18,847 contained control group data and rows 18,848 through 37,988 contained experimental group data. Scores of PCs 1-3 were plotted in two dimensions. PCs 1-3 were selected individually and in combinations for GMM clustering. 2 clusters and 10 iterations were specified.

**Linear Discriminant Analysis for Image Segmentation Results and Discussion**

Image segmentation of sections c-ak-1 through c-ak-5 using PCs 1-3 with LDA was unsuccessful. The data was processed assuming both 2 and 3 classes. Clustering results are shown in Figures 5 through 9. Although c-ak-1 was used to construct the training set, LDA analysis of c-ak-1 did not produce the same clusters that were specified during construction of the training set. The assumption of 3 classes with the use of PC 1 produced results that looked similar to visible light microscope images of samples c-ak-1, c-ak-3, and c-ak-4. This success is in line with later image segmentation results using GMM.

**Linear Discriminant Analysis for Image Segmentation Conclusions**

The challenge of supervised clustering techniques like LDA is that they require a priori knowledge of the data being analyzed. In fact, analysis assuming two classes produced less accurate results because as will be shown later, there are actually three distinct regions in the images. It is also difficult to obtain a representative training set for analysis of unknown data, especially if the two are acquired under different conditions. As with any instrumental technique, both data sets need to be acquired at the same time and under the same conditions.
for accurate comparison. As seen above, classification of the same sample used to generate the training set did not produce accurate results. Furthermore, analysis of the other samples did not always result in clustering.

**Figure 5.** LDA analysis of PC 1 (A, B), PC 2 (C, D), and PC 3 (E, F) for c-ak-1 assuming 2 classes (B, D, F) and 3 classes (A, C, E). Note that LDA classification of (E) did not result in clustering.
Figure 6. LDA analysis of PC 1 (A, B), PC 2 (C, D), and PC 3 (E, F) for c-ak-2 assuming 2 classes (B, D, F) and 3 classes (A, C, E). Note that LDA classification of (E) did not result in clustering.
Figure 7. LDA analysis of PC 1 (A, B), PC 2 (C, D), and PC 3 (E, F) for c-ak-3 assuming 2 classes (B, D, F) and 3 classes (A, C, E). Note that LDA classification of both (C) and (E) did not result in clustering.
Figure 8. LDA analysis of PC 1 (A, B), PC 2 (C, D), and PC 3 (E, F) for c-ak-4 assuming 2 classes (B, D, F) and 3 classes (A, C, E). Note that LDA classification of (D) did not result in clustering.
**Figure 9.** LDA analysis of PC 1 (A, B), PC 2 (C, D), and PC 3 (E, F) for c-ak-5 assuming 2 classes (B, D, F) and 3 classes (A, C, E).

**Gaussian Mixture Model for Image Segmentation Results and Discussion**

Preprocessing with PCA was necessary for data clustering using GMM. Data not preprocessed with PCA was input into the GMM algorithm in the form of entire spectra with absorbances at different wavelengths serving as variables. Table 1 and Table 2 show the
percentage of data sets clustered into the number of specified Gaussian distributions using PCs 1-3 individually and in combinations.

<table>
<thead>
<tr>
<th></th>
<th>w/o PCA</th>
<th>w/ PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>c-ak</td>
<td>83%</td>
<td>100%</td>
</tr>
<tr>
<td>b-ak</td>
<td>50%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Table 1.** Results of GMM clustering with and without preprocessing with PCA and assuming 2 clusters

<table>
<thead>
<tr>
<th></th>
<th>w/o PCA</th>
<th>w/ PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>c-ak</td>
<td>74%</td>
<td>100%</td>
</tr>
<tr>
<td>b-ak</td>
<td>29%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Table 2.** Results of GMM clustering with and without preprocessing with PCA and assuming 3 clusters

For the non-preprocessed data, there is a marked disparity in clustering success between the control and experimental group samples. This may have resulted from different spectral resolutions of the two groups. While the x-bk samples had a spectral resolution of 8 cm\(^{-1}\) like both the c-ak and c-bk samples, the x-ak samples had a spectral resolution of 4 cm\(^{-1}\). Preprocessing with PCA likely provides a degree of normalization for a data set that leads to successful clustering by GMM regardless of the initial physical measurements.

The results of PCs 1-3 selected for input in GMM and assumption of 2 and 3 Gaussian distributions in the data are shown in Figure 10. Selection of PC 1 from data preprocessed with PCA for input into the GMM algorithm and an assumption of 3 Gaussian distributions resulted in the best image segmentation into background and tissue regions. This occurred for all 48 samples belonging to both the control and experimental group. Adding a higher order PC to an input data set containing PC 1 did not noticeably alter the clustering output. This is likely due to
the fact that PC 1 accounts for the greatest variance in the data and any contributions from higher order PCs are minimal.

![Figure 10. GMM analysis of PC 1 (A, E), PC 2 (B, F), PC 3 (C, G), and PC 1+2 (D, H) for c-ak-1 assuming 2 classes (A, B, C, D) and 3 classes (E, F, G, H).]

Examination of visible light microscope images of the samples reveals three regions that correspond to the number of clusters in the segmented data sets. In addition to regions containing tissue, the non-tissue portion of the collected data is made up of regions of empty glass slide and regions of slide covered in paraffin. These areas are distinguished in Figure 11.
Figure 11. Regions of empty slide (A), paraffin (B), and tissue (C) in sample c-ak-1

The average absorbance spectrum of each cluster was computed and plotted in Figure 12 for comparison. There are significant differences in absorbance in regions containing no peaks such as the region between 1800 and 2600 cm\(^{-1}\). These differences may be due to variations in thickness that result in varying degrees of light attenuation.

Examination of the loadings for PC 1 in Figure 13 shows the weighting of absorbances at wavenumbers in the analyzed IR region. Due to the relative success of this method, it was not necessary to examine a different IR band. The peak around 3400 cm\(^{-1}\) had the greatest weight in PC 1. This peak corresponds to an O-H stretch, which indicates that a compound containing a hydroxyl group was a significant source of variance in the samples. Water appears to be a likely candidate for the source of the hydroxyl group because a significant portion of biological matter is composed of water. However, the samples were freeze dried prior to IR analysis, so there should not have been appreciable water content with adequate freeze drying. In addition, the OCT compound used to process the samples contains polyvinyl alcohol and polyethylene glycol, which also contain hydroxyl groups.
Figure 12. Average spectra of 3 clusters corresponding to tissue, slide, and paraffin.
Figure 13. PC 1 loadings for c-ak-1. Note that the peak corresponding to the O-H stretch has the greatest weight.

Because PCs are orthogonal axes in the new reduced data set, they can be plotted against each other to reveal clusters of data points. A plot of PC 2 v. PC 1 scores shown in Figure 14 reveals that boundaries between data points belonging to different parts of the sample are easily visible in this principal component space. Once again, because PC 1 accounts for the greatest variance in the data, plotting it along with PC 2 does not significantly alter clustering results. When scores for PC 3 v. PC 2 are plotted, as shown in Figure 15, the data points corresponding to each region were not distinctly separate as they were in the plot of PC 2 v. PC 1. Examining
these scores plots provides insight into the success of GMM clustering using PC 1 over other PCs.

Figure 14. Scores plot of PC 1 and PC 2. Note the distinct boundaries between clusters.

Figure 15. Scores plot of PC 2 and PC 3. Note the lack of distinct boundaries between clusters.
Gaussian Mixture Model for Image Segmentation Conclusions

GMM as an unsupervised clustering technique used in conjunction with PCA is ideal for image segmentation. It allows individual samples to be analyzed without the need for a training set. Preprocessing with PCA is necessary for reduction and normalization of the data and resulted in clustering 100% of the time. PC 1 can be used to separate data based on the largest variance. The number of Gaussian distributions in the data was three in this case, but it depends on the nature of the samples.

Gaussian Mixture Model for Tissue Analysis Results and Discussion

The average spectra of all control group samples above the kidney branch (c-ak-1 through c-ak-12) and all experimental group samples above the kidney branch (x-ak-3 through x-ak-12) are shown in Figure 16.
Figure 16. Average Spectra for c-ak-1 through c-ak-12 and x-ak-3 through x-ak-12. Note the similarity between these spectra.

The results of GMM cluster analysis using of a concatenated array of control and experimental group tissue using different PCs are shown in Tables 3-6. The array contained samples c-ak-1 through c-ak-12 and x-ak-3 through x-ak-12. Again, the total number of spectra was 37,988. Rows 1 through 18,847 contained control group data and rows 18,848 through 37,988 contained experimental group data. The GMM algorithm output an array of indices, either 1 or 2, indicating the cluster assignment of the input data points. To assess the success of GMM clustering using different PCs, the percentage of data points from a given group that were assigned to one of the clusters was computed. The number of times an index appeared in the rows of the control group data was divided by the number of rows containing control group data. The number of times the other index appeared in the rows of the experimental group data was divided by the number of rows containing experimental group data. When beginning this process, the index appearing most often in the control group rows was selected as the one for success quantification.

<table>
<thead>
<tr>
<th>720-4000 cm⁻¹</th>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC1+2</th>
<th>PC2+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>34%</td>
<td>68%</td>
<td>58%</td>
<td>66%</td>
<td>67%</td>
</tr>
<tr>
<td>AAA</td>
<td>63%</td>
<td>86%</td>
<td>73%</td>
<td>37%</td>
<td>88%</td>
</tr>
<tr>
<td>Average</td>
<td>48%</td>
<td>77%</td>
<td>65%</td>
<td>52%</td>
<td>78%</td>
</tr>
</tbody>
</table>

Table 3. Clustering success for different PCs using the region 720-4000 cm⁻¹

<table>
<thead>
<tr>
<th>720-1504 cm⁻¹</th>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC1+2</th>
<th>PC2+3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthy</td>
<td>64%</td>
<td>64%</td>
<td>73%</td>
<td>59%</td>
<td>65%</td>
</tr>
<tr>
<td>AAA</td>
<td>59%</td>
<td>87%</td>
<td>55%</td>
<td>75%</td>
<td>86%</td>
</tr>
<tr>
<td>Average</td>
<td>62%</td>
<td>75%</td>
<td>64%</td>
<td>67%</td>
<td>75%</td>
</tr>
</tbody>
</table>

Table 4. Clustering success for different PCs using the region 720-1504 cm⁻¹
PC 2 and its combination with PC 3 showed the greatest clustering success. Again, this is because higher order PCs contain less of the variance of the data and thus contribute little when combined with lower order PCs. It is important to note that these results are different from those presented earlier when separating tissue and background data. For separating the tissue from the background PC 1 was the critical PC, but here PC 1 and its combination with PC 2 showed poorer clustering success when compared to PC 2 and PC 3 alone and in combination. This may be explained by the earlier observation that the greatest weighting in PC 1 was in the O-H stretch region and likely resulted from water in the samples. PC 1 in this case may also result in data clustering based on water content, which is likely the greatest source of variance in the samples. Scores plots for PCs 1-3 are shown in Figures 17 and 18. The plot of PC 3 v. PC 2 shows a more defined boundary between the two data sets than the plot of PC 2 v. PC 1. Again, when two PCs are plotted on orthogonal axes, the higher order PC will not have much contribution.
**Figure 17.** Scores plot of PC 1 and PC 2.

**Figure 18.** Scores plot of PC 2 and PC 3. Note the presence of a more distinct boundary between the two clusters than in the figure above.
Success with PC 2 did not differ significantly when region 720-4000 cm$^{-1}$ was analyzed than when regions 720-3104 cm$^{-1}$, 3112-4000 cm$^{-1}$, and 720-1504 cm$^{-1}$ were analyzed. Because only a single mouse was represented in each group, however, more samples need to be acquired to establish an estimate of error. Because PC 2 showed significantly better clustering success than the other PCs, comparing the loadings of PC 2 to those of other PCs can help establish which infrared bands are critical for accurately distinguishing between healthy and diseased samples. Examination of loadings from analysis of region 720-4000 cm$^{-1}$ in Figure 19 shows that the band corresponding to the O-H stretch is similarly weighted in PC 1 and PC 2.

**Figure 19.** Loadings for PCA of region 720-4000 cm$^{-1}$ (complete spectrum). Note the significant weighting of the band above 3600 cm$^{-1}$ and from 1720 to 2720 cm$^{-1}$ in PC 2.
When the data was analyzed in region 3112-4000 cm\(^{-1}\), the loadings shown in Figure 20 also showed a large weighting of the O-H stretch band for PCs 1 and 2. However, the band above 3600 cm\(^{-1}\) was highly weighted in PC 2 but not in PC 1 when the spectra were analyzed in regions 720-4000 cm\(^{-1}\) and 3112-4000 cm\(^{-1}\). The loadings for PC 3 are not shown in Figure 20 because input of PC 3 into the GMM algorithm did not result in any clustering.

![Loadings: 3112–4000 cm\(^{-1}\)](image)

**Figure 20.** Loadings for PCA of region 3112-4000 cm\(^{-1}\) where the O-H stretch band is present. The loadings for PC 3 are not shown because it did not result in clustering when input into the GMM algorithm.

The band from 1720 to 2720 cm\(^{-1}\) was heavily weighted in PC 2 when the spectra were analyzed in region 720-4000 cm\(^{-1}\) and 720-3104 cm\(^{-1}\) shown in Figures 19 and 21. However,
this band was more heavily weighted in PC 1 in Figure 21. Referring back to Figure 16 it is clear that there is a significant difference in absorbance for the band above 3600 cm\(^{-1}\) and from 1720 to 2720 cm\(^{-1}\) between the average spectra for the healthy and diseased samples. These bands, however, do not contain any peaks, so differences in absorbance may be due to variations in thickness of the samples.

![Loadings: 720–3104 cm\(^{-1}\)](image)

**Figure 21.** Loadings for PCA of the fingerprint region

Analysis of the fingerprint region 720-1504 cm\(^{-1}\) showed a greater weighting of the peak at 1112 cm\(^{-1}\) and the band from 1420 to 1504 cm\(^{-1}\) in PC 2. These bands, however, have relatively low weights in PC 2 relative to PCs 1 and 3 when regions 720-4000 cm\(^{-1}\) and 720-3104 cm\(^{-1}\) were analyzed.
Figure 22. Loadings for PCA of the region 720-3104 cm\(^{-1}\) below the O-H stretch band.

**Gaussian Mixture Model for Tissue Analysis Conclusions**

It is important to keep in mind that spectral differences in the control and experimental group samples may not necessarily be a direct result of chemical differences between the tissue samples. Inconsistent processing and analysis of the samples may have affected spectral outputs. 2D FT-IR analysis is usually a time consuming process. During spectral analysis there is a large window of time for shifts in instrument reading to occur, which would result in different average spectra for the control and experimental group samples if they were not analyzed in
random order. This factor may have produced the significant weighting of the band above 3600 cm\(^{-1}\) and from 1720-2720 cm\(^{-1}\) in PC 2. For this reason future samples should be analyzed in random order. In addition, variations in sample thickness, which can cause attenuation or scattering of light, may have also been responsible for spectral differences between the samples. Recall that absorbance is directly related to path length by Beer's law. Although the mouse aortas were sectioned using a cryostat, any minor variations in cuts may have been significant considering the small scale of things. Taking ratios of peaks would cancel out the thickness effects, but which peaks to select would have to be determined. It may be beneficial to eventually remove spectral regions that do not correspond to any peaks. At the same time, in this investigation these regions were useful for identifying issues in the data.

References


(28) FrantzDale, B. *Principal Component Analysis;* 2009.


Appendix – MatLab Functions

Contents

Principal Component Analysis Functions  39
  Function A1 – princomp  39
Gaussian Mixture Model Functions  42
  Function A2 – GMM  42
  Function A3 – kmeans  44
  Function A4 – cluster_probability  59
  Function A5 – cluster_covariance  60
  Function A6 – probability_of_cluster_given_x  61
  Function A7 – compute_mean_for_cluster  62
  Function A8 – probability_of_x  63
  Function A9 – mixing_coefficient  64
Linear Discriminant Analysis Functions  65
  Function A10 – LDA  65
  Function A11 – classificationLDA  67
  Function A12 – classifyLDA  72
Data Processing Functions  74
  Function A13 – tissueSelector  76
  Function A14 – tissueExtractor  76
Principal Component Analysis Functions

Function A1 – princomp

function [coeff, score, latent, tsquare] = princomp(x,econFlag)
% PRINCOMP Principal Components Analysis (PCA) from raw data.
% COEFF = PRINCOMP(X) performs principal components analysis on the N-by-P
% data matrix X, and returns the principal component coefficients, also
% known as loadings. Rows of X correspond to observations, columns to
% variables. COEFF is a P-by-P matrix, each column containing coefficients
% for one principal component. The columns are in order of decreasing
% component variance.
% PRINCOMP centers X by subtracting off column means, but does not
% rescale the columns of X. To perform PCA with standardized variables,
% i.e., based on correlations, use PRINCOMP(ZSCORE(X)). To perform PCA
% directly on a covariance or correlation matrix, use PCACOV.
% [COEFF, SCORE] = PRINCOMP(X) returns the principal component scores,
% i.e., the representation of X in the principal component space. Rows
% of SCORE correspond to observations, columns to components.
% [COEFF, SCORE, LATENT] = PRINCOMP(X) returns the principal component
% variances, i.e., the eigenvalues of the covariance matrix of X, in
% LATENT.
% [COEFF, SCORE, LATENT, TSQUARED] = PRINCOMP(X) returns Hotelling's
% T-squared statistic for each observation in X.
% When N <= P, SCORE(:,N:P) and LATENT(N:P) are necessarily zero, and the
% columns of COEFF(:,N:P) define directions that are orthogonal to X.
% [...] = PRINCOMP(X,'econ') returns only the elements of LATENT that are
% not necessarily zero, i.e., when N <= P, only the first N-1, and the
% corresponding columns of COEFF and SCORE. This can be significantly
% faster when P >> N.
% See also BARTTEST, BIPILOT, CANONCORR, FACTORAN, PCACOV, PCARES,
% ROTATEFACTORS.
% References:
% [1] Jackson, J.E., A User's Guide to Principal Components,
% [3] Krzanowski, W.J., Principles of Multivariate Analysis,
% Copyright 1993-2010 The MathWorks, Inc.
% $Revision: 1.1.8.4 $ $Date: 2011/05/09 01:26:32 $

if nargin < 2, econFlag = 0; end
[n,p] = size(x);
if isempty(x)
  pOrZero = ~isequal(econFlag, 'econ') * p;
  coeff = zeros(p,pOrZero); coeff(1:p+1:end) = 1;
  score = zeros(n,pOrZero);
  latent = zeros(pOrZero,1);
  tsquare = zeros(n,1);
  return
end

% Center X by subtracting off column means
x0 = bsxfun(@minus,x,mean(x,1));

if nargout < 2
  if n >= p && (isequal(econFlag,0) || isequal(econFlag,'econ'))
    % When only coefs are needed, EIG is significantly faster than SVD.
    [coeff,~] = eig(x0'*x0);
    coeff = fliplr(coeff);
  else
    % The principal component coefficients are the eigenvectors of
    % S = X0'*X0./(n-1), but computed using SVD.
    [~,~,coeff] = svd(x0,econFlag);
  end
  % When econFlag is 'econ', only (n-1) components should be returned.
  % See comment below.
  if (n <= p) && isequal(econFlag,'econ')
    coeff(:,n) = [];
  end
else
  r = min(n-1,p); % max possible rank of X0

  % The principal component coefficients are the eigenvectors of
  % S = X0'*X0./(n-1), but computed using SVD.
  [U,sigma,coeff] = svd(x0,econFlag); % put in 1/sqrt(n-1) later

  % Project X0 onto the principal component axes to get the scores.
  if n == 1 % sigma might have only 1 row
    sigma = sigma(1);
  else
    sigma = diag(sigma);
  end
  score = bsxfun(@times,U,sigma'); % == x0*coeff
  sigma = sigma ./ sqrt(n-1);

  % When X has at least as many variables as observations, eigenvalues
  % n:p of S are exactly zero.
  if n <= p
    % When econFlag is 'econ', nothing corresponding to the zero
    % eigenvalues should be returned. svd('econ') won't have
    % returned anything corresponding to components (n+1):p, so we
    % just have to cut off the n-th component.
    if isequal(econFlag, 'econ')
      sigma(n,:) = [];
      coeff(:,n) = [];
      score(:,n) = [];
    else
      sigma = diag(sigma);
    end
  end
% Otherwise, set those eigenvalues and the corresponding scores to
% exactly zero. svd(0) won't have returned columns of U
% corresponding to components (n+1):p, need to fill those out.
else
    sigma(n:p,1) = 0; % make sure this extends as a column
    score(:,n:p) = 0;
end
end

% The variances of the pc's are the eigenvalues of S = X0'*X0./(n-1).
latent = sigma.^2;

% Hotelling's T-squared statistic is the sum of squares of the
% standardized scores, i.e., Mahalanobis distances. When X appears to
% have column rank < r, ignore components that are orthogonal to the
% data.
if nargout == 4
    if n > 1
        q = sum(sigma > max(n,p).*eps(sigma(1)));
        if q < r
            warning(message('stats:princomp:colRankDefX', q));
        end
    else
        q = 0;
    end
    tsquare = (n-1) .* sum(U(:,1:q).^2,2); % ==
    sum((score*diag(1./sigma)).^2,2)
end
end

% Enforce a sign convention on the coefficients -- the largest element in
% each column will have a positive sign.
 [~,maxind] = max(abs(coeff),[],1);
 d = size(coeff,2);
 colsign = sign(coeff(maxind + (0:p:(d-1)*p)));
 coeff = bsxfun(@times,coeff,colsign);
 if nargout > 1
    score = bsxfun(@times,score,colsign);
end
Gaussian Mixture Model Functions

Function A2 – GMM

% Clustering by Gaussian Mixture Model
% Author: T. N. Vikram
% Place: International School of Information Management
% Reference: Algorithm Collections For Digital Signal Processing
% Applications Using Matlab – E.S. Gopi, Kluwer 2007
% Email: vikram@isim.ac.in

% GMM Example
% x1 = 10 + sqrt(3) * randn(5,3);
% x2 = 20 + sqrt(5) * randn(5,3);
% x3 = 25 + sqrt(2) * randn(5,3);
% Input = [x1 x2 x3];
% No_of_Clusters = 2;
% No_of_Iterations = 5;
% [INDEX, Mu, Variances] = GMM(Input, No_of_Clusters, No_of_Iterations);

function [INDEX, Mu, Variances] = GMM(Input, No_of_Clusters, Limit)

% Initialize the Cluster Centroid
[IDX, Initial_Centroids] = kmeans(Input', No_of_Clusters);
Mu = Initial_Centroids';
Limit = 10;
for Iterations = 1:Limit
    [No_of_Features_within_Data, No_of_Data_Points] = size(Input);
    Probability_of_Cluster_given_Point(1:No_of_Clusters, 1:No_of_Data_Points) = 0.0;
    [PC, INDEX] = Cluster_Probability(Input, Mu);

    % Initialize Cluster Covariances
    COVAR(1:No_of_Features_within_Data, 1:No_of_Clusters) = 0.0;
    for i=1:No_of_Clusters
        COVAR(:,i) = Cluster_Covariance(Input(:,IDX==i));
    end

    % Initialize the probability matrix P(Cluster/Point)
    Variances = COVAR;
    for i=1:No_of_Clusters
        for j=1:No_of_Data_Points
            Probability_of_Cluster_given_Point(i,j) = Probability_of_Cluster_given_X(Input(:,j), Mu, Variances, PC, i);
        end
    end

    % New Means
    Mu(1:No_of_Clusters, 1:No_of_Features_within_Data) = 0.0;
    for i=1:No_of_Clusters
        Mu(i,:) = Compute_Mean_for_Cluster(Input, Mu, Variances, PC, i);
    end
end
%disp(Iterations);
\%disp(Mu1);
Mu = Mul';
end;
Function A3 – kmeans

function [idx, C, sumD, D] = kmeans(X, k, varargin)
%KMEANS K-means clustering.
%   IDX = KMEANS(X, K) partitions the points in the N-by-P data matrix X
%   into K clusters. This partition minimizes the sum, over all clusters, of
%   the within-cluster sums of point-to-cluster-centroid distances. Rows of
%   X correspond to points, columns correspond to variables. Note: when X is a
%   vector, KMEANS treats it as an N-by-1 data matrix, regardless of its
%   orientation. KMEANS returns an N-by-1 vector IDX containing the cluster
%   indices of each point. By default, KMEANS uses squared Euclidean
%   distances.
%   KMEANS treats NaNs as missing data, and ignores any rows of X that
%   contain NaNs.
%   [IDX, C] = KMEANS(X, K) returns the K cluster centroid locations in
%   the K-by-P matrix C.
%   [IDX, C, SUMD] = KMEANS(X, K) returns the within-cluster sums of
%   point-to-centroid distances in the 1-by-K vector sumD.
%   [IDX, C, SUMD, D] = KMEANS(X, K) returns distances from each point
%   to every centroid in the N-by-K matrix D.
%   [ ... ] = KMEANS(..., 'PARAM1',val1, 'PARAM2',val2, ...) specifies
%   optional parameter name/value pairs to control the iterative algorithm
%   used by KMEANS. Parameters are:
%   'Distance' - Distance measure, in P-dimensional space, that KMEANS
%     should minimize with respect to. Choices are:
%     'sqEuclidean' - Squared Euclidean distance (the default)
%     'cityblock' - Sum of absolute differences, a.k.a. L1 distance
%     'cosine' - One minus the cosine of the included angle
%              between points (treated as vectors)
%     'correlation' - One minus the sample correlation between points
%                     (treated as sequences of values)
%     'Hamming' - Percentage of bits that differ (only suitable
%                  for binary data)
%   'Start' - Method used to choose initial cluster centroid positions,
%     sometimes known as "seeds". Choices are:
%     'sample' - Select K observations from X at random (the default)
%     'uniform' - Select K points uniformly at random from the range
%                 of X. Not valid for Hamming distance.
%     'cluster' - Perform preliminary clustering phase on random 10%
%                 subsample of X. This preliminary phase is itself
%                 initialized using 'sample'.
%     'matrix' - A K-by-P matrix of starting locations. In this case,
%                 you can pass in [] for K, and KMEANS infers K from
%                 the first dimension of the matrix. You can also
%                 supply a 3D array, implying a value for 'Replicates'
%                 from the array's third dimension.
%   'Replicates' - Number of times to repeat the clustering, each with a
new set of initial centroids. A positive integer, default is 1.

'EmptyAction' - Action to take if a cluster loses all of its member
observations. Choices are:
- 'error' - Treat an empty cluster as an error (the default)
- 'drop' - Remove any clusters that become empty, and set
  the corresponding values in C and D to NaN.
- 'singleton' - Create a new cluster consisting of the one
  observation furthest from its centroid.

'Options' - Options for the iterative algorithm used to minimize the
fitting criterion, as created by STATSET. Choices of STATSET
parameters are:
- 'Display' - Level of display output. Choices are 'off', (the
default), 'iter', and 'final'.
- 'MaxIter' - Maximum number of iterations allowed. Default is
  100.

'OnlinePhase' - Flag indicating whether KMEANS should perform an "on-line
update" phase in addition to a "batch update" phase. The on-line
phase can be time consuming for large data sets, but guarantees a solution
that is a local minimum of the distance criterion, i.e., a partition of
the data where moving any single point to a different cluster
increases the total sum of distances. 'on' (the default) or 'off'.

Example:

X = [randn(20,2)+ones(20,2); randn(20,2)-ones(20,2)];
opts = statset('Display','final');
[cidx, ctrs] = kmeans(X, 2, 'Distance','city', ...
  'Replicates',5, 'Options',opts);
plot(X(cidx==1,:),X(cidx==1,:),'r.', ...
  X(cidx==2,:),X(cidx==2,:), 'b.', ctrs(:,1),ctrs(:,2),'kx');

See also LINKAGE, CLUSTERDATA, SILHOUETTE.

KMEANS uses a two-phase iterative algorithm to minimize the sum of
point-to-centroid distances, summed over all K clusters. The first phase
uses what the literature often describes as "batch" updates, where each
iteration consists of reassigning points to their nearest cluster
centroid, all at once, followed by recalculation of cluster centroids.
This phase occasionally (especially for small data sets) does not
converge to solution that is a local minimum, i.e., a partition of the data where
moving any single point to a different cluster increases the total sum of
distances. Thus, the batch phase be thought of as providing a fast but
potentially only approximate solution as a starting point for the second
phase. The second phase uses what the literature often describes as
"on-line" updates, where points are individually reassigned if doing so
will reduce the sum of distances, and cluster centroids are recomputed
after each reassignment. Each iteration during this second phase
consists of one pass though all the points. The on-line phase will converge to a
% local minimum, although there may be other local minima with lower total
% sum of distances. The problem of finding the global minimum can only be
% solved in general by an exhaustive (or clever, or lucky) choice of
% starting points, but using several replicates with random starting points
% typically results in a solution that is a global minimum.
%
% References:
% Programs, Examples, translated by J. Goldschmidt, Halsted Press,
% New York.

% $Revision: 1.1.10.3 $  $Date: 2011/05/09 01:25:32 $

if nargin < 2
    error(message('stats:kmeans:TooFewInputs'));
end

[~,wasnan,X] = statremovenan(X);
hadNaNs = any(wasnan);
if hadNaNs
    warning(message('stats:kmeans:MissingDataRemoved'));
end

% n points in p dimensional space
[n, p] = size(X);

pnames = { 'distance' 'start' 'replicates' 'emptyaction' 'onlinephase'
          'options'
          'maxiter' 'display'};
dflts = { 'sqeuclidean' 'sample' [] 'error' 'on'
         [] [] []};
[distance,start,reps,emptyact,online,options,maxit,display] ...
    = internal.stats.parseArgs(pnames, dflts, varargin{:});
distNames = { 'sqeuclidean', 'cityblock', 'cosine', 'correlation', 'hamming'};
distance = internal.stats.getParamVal(distance,distNames,'''Distance''');
switch distance
    case 'cosine'
        Xnorm = sqrt(sum(X.^2, 2));
        if any(min(Xnorm) <= eps(max(Xnorm)))
           error(message('stats:kmeans:ZeroDataForCos'));
        end
        X = X ./ Xnorm(:,ones(1,p));
    case 'correlation'
        X = X - repmat(mean(X,2),1,p);
        Xnorm = sqrt(sum(X.^2, 2));
        if any(min(Xnorm) <= eps(max(Xnorm)))
           error(message('stats:kmeans:ConstantDataForCorr'));
        end
        X = X ./ Xnorm(:,ones(1,p));
    case 'hamming'
        if ~all(ismember(X(:,[0 1])))
           error(message('stats:kmeans:NonbinaryDataForHamm'));
        end
end
if ischar(start)
    startNames = {'uniform','sample','cluster'};
    j = find(strncmpi(start,startNames,length(start)));  
    if length(j) > 1
        error(message('stats:kmeans:AmbiguousStart', start));
    elseif isempty(j)
        error(message('stats:kmeans:UnknownStart', start));
    elseif isempty(k)
        error(message('stats:kmeans:MissingK'));
    end
    start = startNames{j};
    if strcmp(start,'uniform')
        if strcmp(distance,'hamming')
            error(message('stats:kmeans:UniformStartForHamm'));
        end
        Xmins = min(X,[],1);  
        Xmaxs = max(X,[],1);
    end
elseif isnumeric(start)
    CC = start;
    start = 'numeric';
    if isempty(k)
        k = size(CC,1);
    elseif k ~= size(CC,1)
        error(message('stats:kmeans:StartBadRowSize'));
    elseif size(CC,2) ~= p
        error(message('stats:kmeans:StartBadColumnSize'));
    end
    if isempty(reps)
        reps = size(CC,3);
    elseif reps ~= size(CC,3)
        error(message('stats:kmeans:StartBadThirdDimSize'));
    end

    % Need to center explicit starting points for 'correlation'.
    (Re)normalization
    % for 'cosine'/'correlation' is done at each iteration.
    if isequal(distance,'correlation')
        CC = CC - repmat(mean(CC,2),[1,p,1]);
    end
else
    error(message('stats:kmeans:InvalidStart'));
end

emptyactNames = {'error','drop','singleton'};
emptyact = internal.stats.getParamVal(emptyact,emptyactNames,'''EmptyAction''');

 [~,online] = internal.stats.getParamVal(online,{'on','off'},''OnlinePhase'');
online = (online==1);

% 'maxiter' and 'display' are grandfathered as separate param name/value pairs
if ~isempty(display)
    options = statset(options,'Display',display);
end
if ~isempty(maxit)
    options = statset(options,'MaxIter',maxit);
end

options = statset(statset('kmeans'), options);
display = find(strncmpi(options.Display, {'off','notify','final','iter'},... 
    length(options.Display))) - 1;
maxit = options.MaxIter;

if ~(isscalar(k) && isnumeric(k) && isreal(k) && k > 0 && (round(k)==k))
    error(message('stats:kmeans:InvalidK'));
    % elseif k == 1
    % this special case works automatically
elseif n < k
    error(message('stats:kmeans:TooManyClusters'));
end
% Assume one replicate
if isempty(reps)
    reps = 1;
end
%
% Done with input argument processing, begin clustering
%
dispfmt = '%6d	%6d	%8d	%12g
';
if online, Del = NaN(n,k); end % reassignment criterion
totsumDBest = Inf;
emptyErrCnt = 0;
for rep = 1:reps
    switch start
        case 'uniform'
        C = unifrnd(Xmins(ones(k,1),:), Xmaxs(ones(k,1),:));
        % For 'cosine' and 'correlation', these are uniform inside a
        subset
        % of the unit hypersphere.  Still need to center them for
        % 'correlation'.  (Re)normalization for 'cosine'/'correlation' is
        % done at each iteration.
        if isequal(distance, 'correlation')
            C = C - repmat(mean(C,2),1,p);
        end
        if isa(X,'single')
            C = single(C);
        end
        case 'sample'
        C = X(randsample(n,k),:);
        if ~isfloat(C)    % X may be logical
            C = double(C);
        end
        case 'cluster'
        Xsubset = X(randsample(n,floor(.1*n)),:);
        [~, C] = kmeans(Xsubset, k, varargin{:}, 'start','sample',
        'replicates',1);
        case 'numeric'
C = CC(:,:,rep);

% Compute the distance from every point to each cluster centroid and the
% initial assignment of points to clusters
D = distfun(X, C, distance, 0, rep, reps);
[d, idx] = min(D, [], 2);
m = accumarray(idx,1,[k,1]);

try
% catch empty cluster errors and move on to next rep

% Begin phase one: batch reassignments
converged = batchUpdate();

% Begin phase two: single reassignments
if online
    converged = onlineUpdate();
end

if ~converged
    warning(message('stats:kmeans:FailedToConverge', maxit, repsMsg(rep, reps )));
end

% Calculate cluster-wise sums of distances
nonempties = find(m>0);
D(:,nonempties) = distfun(X, C(nonempties,:), distance, iter, rep, reps);
d = D((idx-1)*n + (1:n)');
sumD = accumarray(idx,d,[k,1]);
totsumD = sum(sumD);

if display > 1 % 'final' or 'iter'
    fprintf('%d iterations, total sum of distances = %g
',iter,totsumD);
end

% Save the best solution so far
if totsumD < totsumDBest
    totsumDBest = totsumD;
    idxBest = idx;
    Cbest = C;
    sumDBest = sumD;
    if nargout > 3
        Dbest = D;
    end
end

% If an empty cluster error occurred in one of multiple replicates,
catch
% it, warn, and move on to next replicate. Error only when all
replicates
% fail. Rethrow an other kind of error.
catch ME
    if reps == 1 || ~isequal(ME.identifier,'stats:kmeans:EmptyCluster')
        rethrow(ME);
    else

emptyErrCnt = emptyErrCnt + 1;
warning(message('stats:kmeans:EmptyClusterInBatchUpdate', rep, iter));
if emptyErrCnt == reps
    error(message('stats:kmeans:EmptyClusterAllReps'));
end
end % catch
end % replicates

% Return the best solution
idx = idxBest;
C = Cbest;
sumD = sumDBest;
if nargout > 3
    D = Dbest;
end
if hadNaNs
    idx = statinsertnan(wasnan, idx);
end

% function converged = batchUpdate
function converged = batchUpdate
% Every point moved, every cluster will need an update
moved = 1:n;
changed = 1:k;
previdx = zeros(n,1);
prevtotsumD = Inf;

if display > 2 % 'iter'
    fprintf(' iter\t phase\t num\t sum\n');
end

% Begin phase one: batch reassignments
%
iter = 0;
converged = false;
while true
    iter = iter + 1;
    % Calculate the new cluster centroids and counts, and update the % distance from every point to those new cluster centroids
    [C(changed,:), m(changed)] = gcentroids(X, idx, changed, distance);
    D(:,changed) = distfun(X, C(changed,:), distance, iter);
    % Deal with clusters that have just lost all their members
    empties = changed(m(changed) == 0);
    if ~isempty(empties)
        switch emptyact
        case 'remove'
            % Remove the empty clusters from the list
            idx = idx(find(idx ~= changed(m(changed) == 0)));
            C = C(:,setdiff(1:k, changed(m(changed) == 0)));
            m = m(setdiff(1:k, changed(m(changed) == 0)));
            sumD = sumD(setdiff(1:k, changed(m(changed) == 0)));
        case 'collapse'
            % Collapse all the empty clusters into the nearest non-empty cluster
            % This is done by assigning each point to the nearest non-empty cluster
            % and updating the cluster centroids and counts accordingly.
            % The details of this process are complex and require detailed analysis.
            % For brevity, we will not include the code here.
        case 'error'
            % If the number of empty clusters exceeds a certain threshold, raise an error
            % This is done to prevent the algorithm from continuing indefinitely.
            % The details of this process are complex and require detailed analysis.
            % For brevity, we will not include the code here.
        end
        % Update the iteration count and the convergence status
    end
% end if
end % while
end
case 'error'
    error(message('stats:kmeans:EmptyCluster', iter, repsMsg(rep, reps )));

    case 'drop'
        % Remove the empty cluster from any further processing
        D(:,empties) = NaN;
        changed = changed(m(changed) > 0);
        warning(message('stats:kmeans:EmptyCluster', iter, repsMsg(rep, reps )));

    case 'singleton'
        warning(message('stats:kmeans:EmptyCluster', iter, repsMsg(rep, reps )));

    for i = empties
        d = D((idx-1)*n + (1:n)');  % use newly updated distances
        % Find the point furthest away from its current cluster.
        % Take that point out of its cluster and use it to create
        % a new singleton cluster to replace the empty one.
        [~, lonely] = max(d);
        from = idx(lonely);  % taking from this cluster
        if m(from) < 2  % In the very unusual event that the cluster had only
            % one member, pick any other non-singleton point.
            from = find(m>1,1,'first');
            lonely = find(idx==from,1,'first');
        end
        C(i,:) = X(lonely,:);  
        m(i) = 1;
        idx(lonely) = i;
        D(:,i) = distfun(X, C(i,:), distance, iter);
    end

    % Update clusters from which points are taken
    [C(from,:), m(from)] = gcentroids(X, idx, from, distance);
    D(:,from) = distfun(X, C(from,:), distance, iter);
    changed = unique([changed from]);
end
end

% Compute the total sum of distances for the current configuration.
totsumD = sum(D((idx-1)*n + (1:n)'));
% Test for a cycle: if objective is not decreased, back out % the last step and move on to the single update phase
if prevtotsumD <= totsumD
    idx = previdx;
\[ [C(\text{changed,:}), m(\text{changed})] = \text{gcentroids}(X, \text{idx}, \text{changed}, \text{distance}); \]

\[ \text{iter} = \text{iter} - 1; \]
\[ \text{break}; \]
\[ \text{end} \]
\[ \text{if} \ \text{display} > 2 \ % 'iter' \]
\[ \text{fprintf}(\text{dispfmt}, \text{iter}, 1, \text{length}(\text{moved}), \text{totsumD}); \]
\[ \text{end} \]
\[ \text{if} \ \text{iter} >= \text{maxit} \]
\[ \text{break}; \]
\[ \text{end} \]

\% Determine closest cluster for each point and reassign points to clusters
\[ \text{previdx} = \text{idx}; \]
\[ \text{prevtotsumD} = \text{totsumD}; \]
\[ [d, \text{nidx}] = \text{min}(D, [], 2); \]

\% Determine which points moved
\[ \text{moved} = \text{find}(\text{nidx} \neq \text{previdx}); \]
\[ \text{if} \ \sim\text{isempty}(\text{moved}) \]
\[ \% \text{Resolve ties in favor of not moving} \]
\[ \text{moved} = \text{moved}(D((\text{previdx}(\text{moved})-1)^*n + \text{moved}) > d(\text{moved})); \]
\[ \text{end} \]
\[ \text{if} \ \sim\text{isempty}(\text{moved}) \]
\[ \text{converged} = \text{true}; \]
\[ \text{break}; \]
\[ \text{end} \]
\[ \text{idx}(\text{moved}) = \text{nidx}(\text{moved}); \]

\% Find clusters that gained or lost members
\[ \text{changed} = \text{unique}([\text{idx}(\text{moved}); \text{previdx}(\text{moved})])'; \]
\[ \text{end} \% \text{phase one} \]
\[ \text{end} \% \text{nested function} \]

\%---------------------------------------------------------------

\textbf{function} \text{converged} = \text{onlineUpdate} \n
\% Initialize some cluster information prior to phase two
\textbf{switch} \text{distance} \n\% case 'cityblock'
\textbf{case} 'cityblock'
\textbf{Xmid} = \text{zeros}([k,p,2]); \n\textbf{for} i = 1:k \n\% Separate out sorted coords for points in i'th cluster, \textbf{if} \ m(i) > 0 \n\% and save values above and below median, component-wise \n\textbf{Xsorted} = \text{sort}(X(idx==i,:,:),1); \n\text{nn} = \text{floor}(0.5*m(i)); \n\textbf{if} \ \text{mod}(m(i),2) == 0 \n\textbf{Xmid}(i,:,1:2) = \text{Xsorted}([\text{nn}, \text{nn}+1],:); \n\textbf{elseif} \ m(i) > 1 \n\textbf{Xmid}(i,:,1:2) = \text{Xsorted}([\text{nn}, \text{nn}+2],:); \n\textbf{end} \n\text{end} \n\text{end} \textbf{case} \n\textbf{end} \n\textbf{switch} \n\text{end}
else
    Xmid(i,:,1:2) = Xsorted([1, 1,:])';
end
end
case 'hamming'
    Xsum = zeros(k,p);
    for i = 1:k
        if m(i) > 0
            % Sum coords for points in i'th cluster, component-wise
            Xsum(i,:) = sum(X(idx==i,:), 1);
        end
    end
end

% Begin phase two: single reassignments
% changed = find(m' > 0);
lastmoved = 0;
nummoved = 0;
iter1 = iter;
converged = false;
while iter < maxit
    % Calculate distances to each cluster from each point, and the
    % potential change in total sum of errors for adding or removing
    % each point from each cluster. Clusters that have not changed
    % membership need not be updated.
    % Singleton clusters are a special case for the sum of dists
    % calculation. Removing their only point is never best, so the
    % reassignment criterion had better guarantee that a singleton
    % point will stay in its own cluster. Happily, we get
    % Del(i,idx(i)) == 0 automatically for them.
    switch distance
        case 'sqeuclidean'
            for i = changed
                mbars = (idx == i);
                sgn = 1 - 2*mbars; % -1 for members, 1 for nonmembers
                if m(i) == 1
                    sgn(mbars) = 0; % prevent divide-by-zero for singleton mbars
                end
                Del(:,i) = (m(i) ./ (m(i) + sgn)) .* sum((X - C(repmat(i,n,1),:)).^2, 2);
            end
        case 'cityblock'
            for i = changed
                if mod(m(i),2) == 0 % this will never catch singleton clusters
                    ldist = Xmid(repmat(i,n,1,:),1) - X;
                    rdist = X - Xmid(repmat(i,n,1,:),2);
                    mbars = (idx == i);
                    sgn = repmat(1-2*mbars, 1, p); % -1 for members, 1
                    for nonmembers
\[ \text{Del}(:,i) = \sum(\max(0, \max(sgn.*rdist, sgn.*ldist)), 2); \]

else
\[ \text{Del}(:,i) = \sum(\abs(X - \text{C(repmat(i,n,1),:\)}), 2); \]
end
end
case {'cosine', 'correlation'}
\% The points are normalized, centroids are not, so normalize them
normC = sqrt(sum(C.^2, 2));
if any(normC < eps(class(normC))) \% small relative to unit-length data points
error(message('stats:kmeans:ZeroCentroid', iter, repsMsg(rep, reps))); end
\% This can be done without a loop, but the loop saves memory allocations
for i = changed
\% coords with an unequal number of 0s and 1s have a different contribution than coords with an equal number
unequal01 = find(2*Xsum(i,:) ~= m(i));
umequal01 = p - length(unequal01);
mbrs = (idx == i);
Di = abs(X(:,unequal01) - C(repmat(i,n,1),unequal01));
Del(:,i) = (sum(Di, 2) + mbrs*numequal01) / p;
else
Del(:,i) = sum(abs(X - C(repmat(i,n,1),:\)), 2) / p;
end
end
case 'hamming'
for i = changed
\% this will never catch singleton clusters
\% coords with an unequal number of 0s and 1s have a different contribution than coords with an equal number
unequal01 = find(2*Xsum(i,:) ~= m(i));
umequal01 = p - length(unequal01);
mbrs = (idx == i);
Di = abs(X(:,unequal01) - C(repmat(i,n,1),unequal01));
Del(:,i) = (sum(Di, 2) + mbrs*numequal01) / p;
end
determine best possible move, if any, for each point. Next we will pick one from those that actually did move.
previdx = idx;
prevtotsumD = totsumD;
[minDel, nidx] = min(Del, [], 2);
moved = find(previdx == nidx);
if ~isempty(moved)
\% Resolve ties in favor of not moving
moved = moved(Del((previdx(moved)-1)*n + moved) > minDel(moved));
end
if isempty(moved)
    % Count an iteration if phase 2 did nothing at all, or if we're
    % in the middle of a pass through all the points
    if (iter == iter1) || nummoved > 0
        iter = iter + 1;
        if display > 2 % 'iter'
            fprintf(dispfmt,iter,2,nummoved,totsumD);
        end
    end
    converged = true;
    break;
end

% Pick the next move in cyclic order
moved = mod(min(mod(moved - lastmoved - 1, n) + lastmoved), n) + 1;

% If we've gone once through all the points, that's an iteration
if moved <= lastmoved
    iter = iter + 1;
    if display > 2 % 'iter'
        fprintf(dispfmt,iter,2,nummoved,totsumD);
    end
    if iter >= maxit,
        break;
    end
    nummoved = 0;
end
nummoved = nummoved + 1;
lastmoved = moved;

oidx = idx(moved);
nidx = nidx(moved);
totsumD = totsumD + Del(moved,nidx) - Del(moved,oidx);

% Update the cluster index vector, and the old and new cluster
% counts and centroids
idx(moved) = nidx;
m(nidx) = m(nidx) + 1;
m(oidx) = m(oidx) - 1;
switch distance
    case 'sqeuclidean'
        C(nidx,:) = C(nidx,:) + (X(moved,:) - C(nidx,:)) / m(nidx);
        C(oidx,:) = C(oidx,:) - (X(moved,:) - C(oidx,:)) / m(oidx);
    case 'cityblock'
        for i = [oidx nidx]
            Xsorted = sort(X(idx==i,:),1);
            nn = floor(.5*m(i));
            if mod(m(i),2) == 0
                C(i,:) = .5 * (Xsorted(nn,:) + Xsorted(nn+1,:));
            else
                C(i,:) = .5 * (Xsorted(nn,:) + Xsorted(nn+1,:) + Xsorted(nn-1,:));
            end
        end
    end
Xmid(i,:,1:2) = Xsorted([nn, nn+1,:])';
else
    C(i,:) = Xsorted(nn+1,:);
    if m(i) > 1
        Xmid(i,:,1:2) = Xsorted([nn, nn+2,:])';
    else
        Xmid(i,:,1:2) = Xsorted([1, 1,:])';
    end
end
end

% Update summed coords for points in each cluster. New
% centroid is the coord median. All done component-wise.
Xsum(nidx,:) = Xsum(nidx,:) + X(moved,:);  
Xsum(oidx,:) = Xsum(oidx,:) - X(moved,:);  
C(nidx,:) = .5*sign(2*Xsum(nidx,:) - m(nidx)) + .5;  
C(oidx,:) = .5*sign(2*Xsum(oidx,:) - m(oidx)) + .5;
end

% phase two
changed = sort([oidx nidx]);
end
end % nested function
end % main function

function D = distfun(X, C, dist, iter, rep, reps)
%DISTFUN Calculate point to cluster centroid distances.
[n,p] = size(X);
D = zeros(n, size(C,1));
nclusts = size(C,1);

switch dist
    case 'sqeuclidean'
        for i = 1:nclusts
            D(:,i) = (X(:,1) - C(i,1)).^2;
            for j = 2:p
                D(:,i) = D(:,i) + (X(:,j) - C(i,j)).^2;
            end
            % D(:,i) = sum((X - C(repmat(i,n,1),:)).^2, 2);
        end
    end
    case 'cityblock'
        for i = 1:nclusts
            D(:,i) = abs(X(:,1) - C(i,1));
            for j = 2:p
                D(:,i) = D(:,i) + abs(X(:,j) - C(i,j));
            end
            % D(:,i) = sum(abs(X - C(repmat(i,n,1),:))), 2);
        end
    end
    case {'cosine','correlation'}
        % The points are normalized, centroids are not, so normalize them

end

%--

%------------------------------------------------------------------
%------------------------------------------------------------------

function D = distfun(X, C, dist, iter, rep, reps)
%DISTFUN Calculate point to cluster centroid distances.
[n,p] = size(X);
D = zeros(n, size(C,1));
nclusts = size(C,1);

switch dist
    case 'sqeuclidean'
        for i = 1:nclusts
            D(:,i) = (X(:,1) - C(i,1)).^2;
            for j = 2:p
                D(:,i) = D(:,i) + (X(:,j) - C(i,j)).^2;
            end
            % D(:,i) = sum((X - C(repmat(i,n,1),:)).^2, 2);
        end
    end
    case 'cityblock'
        for i = 1:nclusts
            D(:,i) = abs(X(:,1) - C(i,1));
            for j = 2:p
                D(:,i) = D(:,i) + abs(X(:,j) - C(i,j));
            end
            % D(:,i) = sum(abs(X - C(repmat(i,n,1),:))), 2);
        end
    end
    case {'cosine','correlation'}
        % The points are normalized, centroids are not, so normalize them

end

%--
normC = sqrt(sum(C.^2, 2));
if any(normC < eps(class(normC))) % small relative to unit-length
data points
    error(message('stats:kmeans:ZeroCentroid', iter, repsMsg( rep,
    reps )));
    % 'Zero cluster centroid created at iteration %d.'\',iter);
end
for i = 1:nclusts
    D(:,i) = max(1 - X * (C(i,:)./normC(i))', 0);
end
for i = 1:nclusts
    D(:,i) = abs(X(:,1) - C(i,1));
    for j = 2:p
        D(:,i) = D(:,i) + abs(X(:,j) - C(i,j));
    end
    D(:,i) = D(:,i) / p;
    % D(:,i) = sum(abs(X - C(repmat(i,n,1),:)), 2) / p;
end

case 'hamming'
    for i = 1:nclusts
        D(:,i) = abs(X(:,1) - C(i,1));
        for j = 2:p
            D(:,i) = D(:,i) + abs(X(:,j) - C(i,j));
        end
        D(:,i) = D(:,i) / p;
        % D(:,i) = sum(abs(X - C(repmat(i,n,1),:)), 2) / p;
    end
end
end % function
%------------------------------------------------------------------

function [centroids, counts] = gcentroids(X, index, clusts, dist)
%GCENTROIDS Centroid and counts stratified by group.
P = size(X,2);
num = length(clusts);
centroids = NaN(num,p);
counts = zeros(num,1);
for i = 1:num
    members = (index == clusts(i));
    if any(members)
        counts(i) = sum(members);
        switch dist
            case 'squeuclidean'
                centroids(i,:) = sum(X(members,:),1) / counts(i);
            case 'cityblock'
                % Separate out sorted coords for points in i'th cluster,
                % and use to compute a fast median, component-wise
                Xsorted = sort(X(members,:),1);
                nn = floor(.5*counts(i));
                if mod(counts(i),2) == 0
                    centroids(i,:) = .5 * (Xsorted(nn,:) + Xsorted(nn+1,:));
                else
                    centroids(i,:) = Xsorted(nn+1,:);
                end
            case {'cosine','correlation'}
                centroids(i,:) = sum(X(members,:),1) / counts(i); % unnormalized
            case 'hamming'
                % Compute a fast median for binary data, component-wise
                centroids(i,:) = .5*sign(2*sum(X(members,:), 1) - counts(i))
                + .5;
        end
    end
end
function s = repsMsg(rep,reps)
% Utility for warning and error messages.
if reps == 1
    s = ''; 
else 
    s = sprintf(' during replicate %d',rep);
end 
end % function
% Function to compute the Cluster Probability
function [PC,INDEX] = Cluster_Probability(Data,Mu)
[No_of_Features_within_Data,No_of_Data_Points] = size(Data);
[No_of_Features_within_Mu,No_of_Mu_Points] = size(Mu);
PC(1:No_of_Mu_Points) = 0;
INDEX(1:No_of_Data_Points) = 0;
Distance(1:No_of_Data_Points,1:No_of_Mu_Points) = 0.0;

for i=1:No_of_Data_Points
    for j = 1:No_of_Mu_Points
        Distance(i,j) = sqrt(dot(Data(:,i)-Mu(:,j),Data(:,i)-Mu(:,j)));
    end
end

for i=1:No_of_Data_Points
    [value,idx] = min(Distance(i,:));
    PC(idx) = PC(idx)+1;
    INDEX(i) = idx;
end

PC = PC/No_of_Data_Points;
Function A5 – cluster_covariance

% Co-variance Computation for a Cluster
function [COVAR] = Cluster_Covariance(Data)
[r,c] = size(Data);
for i=1:r
    COVAR(i) = var(Data(i,:));
end;
end;
Function A6 – probability_of_cluster_given_x

```matlab
function PY = Probability_of_Cluster_given_X(X, Means, Variances, PC, Label_of_Cluster)
PY = PY / Probability_of_X(X, Means, Variances, PC);
```
Function A7 – compute_mean_for_cluster

function mu = Compute_Mean_for_Cluster(X,Means,Variances,PC,Label_of_Cluster)
[r,c] = size(X);
mu = 0.0;
Numerator = 0.0;
Denominator = 0.0;
for i=1:c
    Numerator = Numerator + Probability_of_Cluster_given_X(X(:,i),Means,Variances,PC,Label_of_Cluster)*X(:,i);
    Denominator = Denominator + Probability_of_Cluster_given_X(X(:,i),Means,Variances,PC,Label_of_Cluster);
end;
mu = Numerator/Denominator;
**Function A8 – probability_of_x**

```matlab
function PX = Probability_of_X(X, Means, Variances, PC)
PX = 0.0;
[r, c] = size(Means);
for i = 1:c
    PX = PX + PC(i) * Mixing_Coefficient(X, Means(:, i), Variances(:, i));
end;
```
function MC = Mixing_Coefficient(X,MU,SIGMA)
    M = normpdf(X,MU,SIGMA);
    [r,c] = size(M);
    MC = 0.0;
    for i=1:c
        MC = MC + M(i);
    end
    MC = MC/c;
Function A10 — LDA

% LDA - MATLAB subroutine to perform linear discriminant analysis
% by Will Dwinnell and Deniz Sevis
% Use:
% W = LDA(Input,Target,Priors)
% W = discovered linear coefficients (first column is the constants)
% Input = predictor data (variables in columns, observations in rows)
% Target = target variable (class labels)
% Priors = vector of prior probabilities (optional)
% Note: discriminant coefficients are stored in W in the order of unique(Target)

%% Example:
%% Generate example data: 2 groups, of 10 and 15, respectively
%% X = [randn(10,2); randn(15,2) + 1.5]; Y = [zeros(10,1); ones(15,1)];
%% Calculate linear discriminant coefficients
%% W = LDA(X,Y);
%% Calculate linear scores for training data
%% L = [ones(25,1) X] * W';
%% Calculate class probabilities
%% P = exp(L) ./ repmat(sum(exp(L),2),[1 2]);
% Last modified: Dec-11-2010

function W = LDA(Input,Target,Priors)

% Determine size of input data
[n m] = size(Input);

% Discover and count unique class labels
ClassLabel = unique(Target);
k = length(ClassLabel);

% Initialize
nGroup = NaN(k,1); % Group counts
GroupMean = NaN(k,m); % Group sample means
PooledCov = zeros(m,m); % Pooled covariance
W = NaN(k,m+1); % model coefficients

if (nargin >= 3) PriorProb = Priors; end

% Loop over classes to perform intermediate calculations
for i = 1:k,
    % Establish location and size of each class
    Group = (Target == ClassLabel(i));
    nGroup(i) = sum(double(Group));
end
% Calculate group mean vectors
GroupMean(i,:) = mean(Input(Group,:));

% Accumulate pooled covariance information
PooledCov = PooledCov + ((nGroup(i) - 1) / (n - k) ).* cov(Input(Group,:));
end

% Assign prior probabilities
if (nargin >= 3)
  % Use the user-supplied priors
  PriorProb = Priors;
else
  % Use the sample probabilities
  PriorProb = nGroup / n;
end

% Loop over classes to calculate linear discriminant coefficients
for i = 1:k,
  % Intermediate calculation for efficiency
  % This replaces:  GroupMean(g,:) * inv(PooledCov)
  Temp = GroupMean(i,:) / PooledCov;

  % Constant
  W(i,1) = -0.5 * Temp * GroupMean(i,:)’ + log(PriorProb(i));

  % Linear
  W(i,2:end) = Temp;
end

% Housekeeping
clear Temp
end

% EOF
Function A11 – classificationLDA

function varargout = classificationLDA(varargin)
% CLASSIFICATIONLDA MATLAB code for classificationLDA.fig
% CLASSIFICATIONLDA, by itself, creates a new CLASSIFICATIONLDA or
% raises the existing
% singleton*.
% H = CLASSIFICATIONLDA returns the handle to a new CLASSIFICATIONLDA or
% the handle to
% the existing singleton*.
% CLASSIFICATIONLDA('CALLBACK',hObject eventdata handles,...) calls the
local
% function named CALLBACK in CLASSIFICATIONLDA.fig with the given input
arguments.
% CLASSIFICATIONLDA('Property','Value',...) creates a new
CLASSIFICATIONLDA or raises the
% existing singleton*. Starting from the left, property value pairs are
applied to the GUI before classificationLDA_OpeningFcn gets called.
An
% unrecognized property name or invalid value makes property application
stop. All inputs are passed to classificationLDA_OpeningFcn via
varargin.
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIDATA
% Edit the above text to modify the response to help classificationLDA
% Last Modified by GUIDE v2.5 22-Jun-2012 13:23:33
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @classificationLDA_OpeningFcn, ...
    'gui_OutputFcn', @classificationLDA_OutputFcn, ...
    'gui_LayoutFcn', [], ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before classificationLDA is made visible.
function classificationLDA_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to classificationLDA (see VARARGIN)

% Choose default command line output for classificationLDA
handles.output = hObject;

set(hObject, 'toolbar', 'figure');

% Update handles structure
guida(hObject, handles);

% UIWAIT makes classificationLDA wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = classificationLDA_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in classify.
function classify_Callback(hObject, eventdata, handles)
% hObject    handle to classify (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

widthTraining=getappdata(0,'iwTraining');
heightTraining=getappdata(0,'ihTraining');
widthTest=getappdata(0,'iwTest');
heightTest=getappdata(0,'ihTest');
selections=get(handles.selections,'Value');
selections=selections';
trainingAbsorbances=getappdata(0,'trainingAbsorbances');
testAbsorbances=getappdata(0,'testAbsorbances');
imagedataTraining=getappdata(0,'imagedataTraining');
imagedataTest=getappdata(0,'imagedataTest');
shift=min(min(trainingAbsorbances));%beginning of scatter correction code
trainingAbsorbances=trainingAbsorbances-shift;
prepdatahandle=@prepdata;%end of scatter correction code
training=prepdata(trainingAbsorbances);%end of scatter correction code
trainingAbsorbances=training(1:length(training),1:132);
tissueSelectorhandle=@tissueSelector;
[X indices Y]=tissueSelectorhandle(trainingAbsorbances,imagedataTraining,10000);
for i=1:1:length(selections)
    Xpcs=X(:,selections(i));

if i==1
    selectedX=Xpcs;
else
    selectedX=[selectedX,Xpcs];
end
LDAhandle=@LDA;
W=LDAhandle(selectedX,Y);
assignin('base','W',W);
classifyLDAhandle=@classifyLDA;
%shift=min(min(testAbsorbances));%beginning of scatter correction code
%testAbsorbances=testAbsorbances-shift;
%prepdatahandle=@prepdata;
%testAbs=testAbsorbances; %end of scatter correction code
%testAbsorbances=testAbs(1:length(testAbs),1:132);
classes=classifyLDAhandle(testAbsorbances,W,selections);
image=reshape(c1asses,heightTest,widthTest);
image=image';
axes(handles.axes)
imagesc(image);
colormap('jet');
set(handles.axes,'DataAspectRatio',[1,1,1]);
set(handles.axes,'PlotBoxAspectRatio',[1,1,1]);
guidata(hObject, handles)

function trainingSetLabel_Callback(hObject, eventdata, handles)
% hObject    handle to trainingSetLabel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of trainingSetLabel as text
%        str2double(get(hObject,'String')) returns contents of
%        trainingSetLabel as a double

% --- Executes during object creation, after setting all properties.
function trainingSetLabel_CreateFcn(hObject, eventdata, handles)
% hObject    handle to trainingSetLabel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get($,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in trainingSetImport.
function trainingSetImport_Callback(hObject, eventdata, handles)
% hObject    handle to trainingSetImport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[nameTraining direcTraining]=uigetfile('*.mat');
if nameTraining==0
    return
end
load(fullfile(direcTraining,nameTraining));
trainingAbsorbances=data;
%trainingAbsorbances=-log10(data/100);
%newReshaped=reshaped(:,,:,1:132);
imagedataTraining=sum(reshaped,3);
[widthTraining heightTraining]=size(imagedataTraining);
setappdata(0,'iwTraining',widthTraining);
setappdata(0,'ihTraining',heightTraining);
setappdata(0,'imagedataTraining',imagedataTraining);
setappdata(0,'trainingAbsorbances',trainingAbsorbances);
set(handles.trainingSetLabel,'String',nameTraining);

% --- Executes on selection change in selections.
function selections_Callback(hObject, eventdata, handles)
    % hObject    handle to selections (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: contents = cellstr(get(hObject,'String')) returns selections
    contents as cell array
    %    contents{get(hObject,'Value')} returns selected item from selections

% --- Executes during object creation, after setting all properties.
function selections_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to selections (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: listbox controls usually have a white background on Windows.
    % See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

% --- Executes on button press in testSetImport.
function testSetImport_Callback(hObject, eventdata, handles)
    % hObject    handle to testSetImport (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    [nameTest direcTest]=uigetfile('*.mat');
    if nameTest==0
        return
    end
load(fullfile(direcTest,nameTest));
%testAbsorbances=-log10(data/100);
testAbsorbances=data;
setappdata(0,'testAbsorbances',testAbsorbances);
newReshaped = reshaped(:, :, 1:132);
imagedataTest = sum(newReshaped, 3);
imagedataTest = sum(reshaped, 3);
[widthTest heightTest] = size(imagedataTest);
setappdata(0, 'iwTest', widthTest);
setappdata(0, 'ihTest', heightTest);
setappdata(0, 'imagedataTest', imagedataTest);
set(handles.testSetLabel, 'String', nameTest);

function testSetLabel_Callback(hObject, eventdata, handles)
    % hObject    handle to testSetLabel (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    %
    % Hints: get(hObject,'String') returns contents of testSetLabel as text
    %        str2double(get(hObject,'String')) returns contents of testSetLabel
    % as a double
    %
    % --- Executes during object creation, after setting all properties.
    function testSetLabel_CreateFcn(hObject, eventdata, handles)
        % hObject    handle to testSetLabel (see GCBO)
        % eventdata  reserved - to be defined in a future version of MATLAB
        % handles    empty - handles not created until after all CreateFcns called
        %
        % Hint: edit controls usually have a white background on Windows.
        %      See ISPC and COMPUTER.
        if ispc && isequal(get(hObject,'BackgroundColor'),...
            get(0,'defaultUicontrolBackgroundColor'))
            set(hObject,'BackgroundColor','white');
        end
Function A12 – classifyLDA

```matlab
function classes = classifyLDA(data, W, selections)
%function image=classifyLDA(data,imagedata,cutoff)

%[tissue,background,indices1,indices2]=tissueSelector(data,imagedata,cutoff);
%select which pixels belong to background and tissue classes based on total transmittance
%tissueArray=length(tissue);%number of tissue pixels
%backgroundArray=length(background);%number of background pixels
%tissueClass=zeros(tissueArray,1);%tissue class labels
%backgroundClass=ones(backgroundArray,1);%background class labels
%Y=[tissueClass;backgroundClass];%class labels, tissue labels first
%X=[tissue;background];%training data
%W=LDA(X,Y);%linear discriminant coefficients
%indices=[indices1;indices2];%indices, tissue indices first
%combo=[indices,P(:,1)];%indices and probabilities of belonging to first class
%sortedIndices=sortrows(combo,1);%sort based on indices
%image=sortedIndices(:,2);
%assignin('base','image',image);
[height width]=size(data);
scores=optPCA(data,width);
for i=1:length(selections)
    score=scores(:,selections(i));
    if i==1
        selectedScores=score;
    else
        selectedScores=[selectedScores,score];
    end
end

for i=1:length(selections)
    class1term=W(1,i)*selectedScores(:,i);
    class2term=W(2,i)*selectedScores(:,i);
    if i==1
        class1terms=class1term;
        class2terms=class2term;
    else
        class1terms=[class1terms,class1term];
        class2terms=[class2terms,class2term];
    end
end
class1sums=sum(class1terms,2);
class2sums=sum(class2terms,2);
classes=zeros(length(data),1);
for i=1:length(data)
    if class1sums(i,1)>=class2sums(i,1)
        classes(i,1)=1;
    end
end
end
% LDA - MATLAB subroutine to perform linear discriminant analysis
% by Will Dwinnell and Deniz Sevis
```
% Use:
% W = LDA(Input,Target,Priors)
% % W       = discovered linear coefficients (first column is the constants)
% % Input   = predictor data (variables in columns, observations in rows)
% % Target  = target variable (class labels)
% % Priors  = vector of prior probabilities (optional)
% % Note: discriminant coefficients are stored in W in the order of 
% unique(Target)
% %
% Example:
% % Generate example data: 2 groups, of 10 and 15, respectively
% X = [randn(10,2); randn(15,2) + 1.5];  Y = [zeros(10,1); ones(15,1)];
% % Calculate linear discriminant coefficients
% W = LDA(X,Y);
% % Calculate linear scores for training data
% L = [ones(25,1) X] * W';
% % Calculate class probabilities
% P = exp(L) ./ repmat(sum(exp(L),2),[1 2]);
% %
% Last modified: Dec-11-2010
**Data Processing Functions**

**Function A13 – tissueSelector**

```matlab
function [ X,indices,Y ] = tissueSelector( absorbances,imagedata,cutoff )
%tissueSelector records the indices of pixels with total absorbance <=
%25000; these pixels correspond to regions of the tissue slice
%make sure imagedata=sum(reshaped,3) without any rotations

[height,width]=size(imagedata);
[points timepoints]=size(absorbances);
arraylength=height*width;
indices1=zeros(arraylength,1);
indices2=zeros(arraylength,1);
outline1=zeros(arraylength,1);
counter1=0;
counter2=0;
scores=optPCA(absorbances,timepoints);

%convert imagedata into single dimension array for use
for i=1:1:height
    for j=1:1:width
        k=((i-1)*width)+j;
        array(k,1)=imagedata(i,j);
    end
end

%tissue data points
for m=1:1:arraylength
    value=array(m,1);
    if value <= cutoff
        indices1(m,1)=m;
        outline1(m,1)=1;
        counter1=counter1+1;
    end
end
outline1=reshape(outline1,width,height);
indices1=indices1(indices1~=0);
tissue=zeros(counter1,timepoints);
for i=1:1:counter1
    index1=indices1(i,1);
    tissue(i,:) = scores(index1,:);
end
assignin('base','tissue');

%Background data points
for i=1:1:arraylength
    value=array(i,1);
    if value > cutoff
        indices2(i,1)=i;
        counter2=counter2+1;
    end
end
```
end

indices2=indices2(indices2~=0);
background=zeros(counter2,timepoints);
for i=1:1:counter2
    index2=indices2(i,1);
    background(i,:)=scores(index2,:);
end
X=[tissue;background];
indices=[indices1;indices2];
tissueLabel=zeros(length(tissue),1);
backgroundLabel=ones(length(background),1);
Y=[tissueLabel;backgroundLabel];

assignin('base','X',X);
assignin('base','indices',indices);
assignin('base','Y',Y);
%assignin('base','outline1',outline1);
assignin('base','tissue',tissue);
assignin('base','background',background);
end
% Function A14 – tissueExtractor

function tissueExtractor( data, PC )
% TISSUEEXTRACTOR Summary of this function goes here
% Detailed explanation goes here

data=-log10(data/100);
[m n]=size(data);
scores=optPCA(data,n);
input=scores(:,1)';
output=GMM(input,3);
output=output';
cluster1=zeros(length(data),1);
cluster2=zeros(length(data),1);
cluster3=zeros(length(data),1);
for i=1:length(data)
    if output(i,1)==1;
        cluster1(i,1)=i;
    else if output(i,1)==2;
        cluster2(i,1)=i;
    else if output(i,1)==3;
        cluster3(i,1)=i;
    end
end
cluster1=cluster1(cluster1~=0);
cluster2=cluster2(cluster2~=0);
cluster3=cluster3(cluster3~=0);
tissue=zeros(length(cluster2),n);
for i=1:length(cluster2)
    tissue(i,:)=data(cluster2(i,1),:);
end
scores=optPCA(tissue,n);
tissue=[cluster2, scores(:,PC)];
assignin('base','tissue',tissue);

% assignin('base','cluster1',cluster1);
% assignin('base','cluster2',cluster2);
% assignin('base','cluster3',cluster3);
end